



Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO FIN DE CARRERA

AUTOMATIZACIÓN Y CONTROL DE UN MOTOR A PASOS PARA LA SINTONIZACIÓN DE FILTROS DE MICROONDAS

(AUTOMATION AND CONTROL OF A STEPPER
MOTOR FOR MICROWAVE FILTER TUNING)

Estudios: Ingeniería de Telecomunicaciones

Autor: Josep Benet Freissinier

Dir.: Zabdiel Brito Brito e Ignacio Llamas Garro

Fecha: Diciembre de 2010

Colaboraciones

Departament de Teoria del Senyal i Comunicacions de la Universitat Politècnica de Catalunya.



Centre Tecnològic de Telecomunicacions de Catalunya.



Este trabajo se engloba dentro del proyecto PIB2010BZ-00585 del Ministerio de Ciencia e Innovación.



Agradecimientos

En primer lugar me gustaría darles las gracias a mis padres por todo el apoyo que me han dado y agradecerles el esfuerzo que han realizado para que todo esto sea posible. A mi hermana, por todos los consejos que me ha dado durante toda la carrera. Y a mis amigos, especialmente a Jose y Eloy, por la ayuda y conocimientos prestados para la realización de este proyecto.

A los profesores implicados en este proyecto, Rosa, Ignacio y en especial a Zabdiel. Ellos han sido los que me han dado la oportunidad de realizarlo y han delegado en mí la responsabilidad de elegir el camino que debía seguir la implementación y el desarrollo de este proyecto.

A los chicos del laboratorio de microondas, que me prestaron su ayuda para hacer la placa hardware y pusieron a mi disposición el material necesario para realizarla.

A Aline por prestarme el resonador y ayudarme en el laboratorio con las mediciones junto con Adrián.

Y a todos aquellos que de forma directa o indirecta han hecho posible este proyecto.

A todos ellos, muchas gracias.

Resumen

Este documento contiene la memoria y resultados experimentales de la realización de un sistema para el autoajuste de la frecuencia central de un resonador de microondas mediante un motor a pasos. El proyecto completo consta de cuatro bloques bien diferenciados:

- 1 La realización del hardware que actúa sobre el motor a pasos.
- 2 El diseño del firmware ubicado en el hardware.
- 3 El software que controla todo el proceso de autoajuste del resonador y obtiene las medidas del analizador de redes.
- 4 La estructura de soporte para ubicar el hardware y posicionar el motor a pasos encima del resonador y encajarlo con el tornillo de ajuste.

El objetivo del proyecto es ajustar de forma automática la frecuencia central de un resonador de microondas a la frecuencia que desee el usuario. Para ello, se introduce la frecuencia deseada en el software de control diseñado en *Matlab* juntamente con otros parámetros necesarios para el sistema. Entonces el software de control manda las órdenes al hardware para que éste actúe sobre las fases del motor haciéndolo girar de tal manera que acabe centrado el resonador a la frecuencia deseada por el usuario. El eje del motor está unido al tornillo de sintonización del resonador y de esta forma, al girar el rotor del motor se hace subir o bajar la frecuencia central del resonador.

Para medir la frecuencia central del resonador, se dispone de un analizador de redes que se comunica con el software de control por GPIB y le envía los parámetros S de la respuesta frecuencial del resonador. Ya en el software se calcula la frecuencia central y se toma la decisión de subir o bajar de frecuencia para ir acercándose a la frecuencia deseada por el usuario.

Para dotar de autonomía al hardware y poder comunicarse vía puerto serie con el PC que contiene el software de control, se ha insertado un microcontrolador que es el encargado de recibir las órdenes del software de control y actuar sobre el motor en función del estado de las variables internas y externas. Obviamente el microcontrolador tiene un firmware diseñado específicamente para tal propósito.

Para todo ello se han utilizado los siguientes programas: *Matlab* para el desarrollo del software de control, *Altium Designer* para el diseño del hardware y *MPLAB* para diseñar el firmware y programar el microcontrolador ubicado en el hardware.

En el capítulo 5 se encuentran los resultados experimentales de algunas sintonizaciones con sus gráficas y su correspondiente análisis.

Resum

Aquest document conté la memòria y els resultats experimentals de la realització d'un sistema per la sintonització de la freqüència central d'un ressonador de microones mitjançant un motor a passos. El projecte complet consta de quatre blocs ben diferenciats:

- 1 La realització d'un hardware que actua sobre el motor a passos.
- 2 El disseny d'un firmware ubicat al hardware.
- 3 El software que controla tot el procés d'ajust del ressonador i obté les mesures de l'analitzador de xarxes.
- 4 L'estructura de suport per ubicar el hardware i posicionar el motor a passos sobre el ressonador i encaixar el cargol d'ajust del filtre a l'eix del motor.

L'objectiu del projecte és ajustar de forma automàtica la freqüència central de un ressonador de microones a la freqüència que desitgi l'usuari. Per aconseguir-ho, s'introdueix la freqüència desitjada al software de control dissenyat en *Matlab* juntament amb altres paràmetres necessaris per al sistema. Llavors, el software de control envia les ordres al hardware per a que aquest actuï sobre les fases del motor fent-lo girar de tal manera que acabi centrant el ressonador a la freqüència desitjada. L'eix del motor està unit al cargol de sintonització del ressonador i d'aquesta manera, al girar el rotor del motor es fa pujar o baixar la freqüència central del ressonador.

Per mesurar la freqüència central del ressonador, es disposa d'un analitzador de xarxes que es comunica amb el software de control mitjançant el bus GPIB i envia els paràmetres S de la resposta freqüencial del ressonador. Ja al software de control es calcula la freqüència central i es pren la decisió de pujar o baixar de freqüència per anar apropant-se a la freqüència desitjada per l'usuari.

Per dotar d'autonomia al hardware i poder comunicar-se per port sèrie amb el PC que conté el software de control, s'ha introduït un microcontrolador que s'encarrega de rebre les ordres del software de control i actuar sobre el motor en funció de l'estat de les variables internes i externes. Naturalment el microcontrolador té un firmware dissenyat específicament per aquest propòsit.

Per al desenvolupament de totes aquestes tasques, s'han utilitzat els següents programes: *Matlab* pel disseny del software de control, *Altium Designer* pel disseny del hardware i *MPLAB* per dissenyar el firmware i programar el microcontrolador del hardware.

Al capítol 5 es troben els resultats experimentals d'algunes sintonitzacions amb les seves gràfiques i anàlisis corresponents.

Abstract

This document contains the report including experimental results of a system implementation for center frequency tuning of a microwave resonator through a stepper motor. The entire project consists of four main blocks:

- 1 Hardware implementation that controls the stepper motor.
- 2 Firmware designs hosted in hardware.
- 3 Control Software for resonator tuning and network analyzer measurement extraction.
- 4 Development of mechanical structure to locate the hardware and hold the stepper motor to fit the resonator tuning screw.

The project aims to adjust the centre frequency of a microwave resonator automatically. Further work consists of implementing multi-resonator tuning, thus filter automatic reconfiguration according to user defined specifications. To do this, the user enters the required frequency in the *Matlab* control software together with other parameters. Then the control software sends orders to the hardware to operate stepper motor phases and rotate the tuning screw above the microwave resonator to achieve center frequency tuning.

To measure the center frequency of the resonator, a network analyzer communicates with the control software using a GPIB bus, recording S parameters of the resonator frequency response. The control software calculates the center frequency and takes the decision to increase or decrease the resonant frequency to match the user defined specification.

A microcontroller has been employed to give autonomy to the hardware and to communicate via a serial port with the PC running the control software. The microcontroller is responsible for receiving orders and acting on the stepper motor phases based on the status of internal and external variables. Obviously the microcontroller has a firmware designed specifically for this purpose.

The following programs have been used in this project: *Matlab* for the control software, *Altium Designer* for hardware design and *MPLAB* for firmware and microcontroller programming hardware-based software development.

Experimental results using the implemented tuning device are discussed in chapter 5, where successful resonator frequency tuning has been achieved.

Índice

<i>Colaboraciones</i>	<i>i</i>
<i>Agradecimientos</i>	<i>iii</i>
<i>Resumen</i>	<i>v</i>
<i>Resum</i>	<i>vii</i>
<i>Abstract</i>	<i>ix</i>
<i>Índice</i>	<i>xi</i>
<i>Lista de acrónimos</i>	<i>xv</i>
<i>Índice de figuras</i>	<i>xvii</i>
<i>Índice de tablas</i>	<i>xix</i>
1. Introducción	1
1.1. Introducción	1
1.2. Motivación y contexto del proyecto	2
1.3. Objetivo del proyecto	3
1.4. Organización de la memoria	5
2. Fundamentos teóricos	7
2.1. El motor a pasos	7
2.1.1. El motor eléctrico de corriente continua	7
2.1.2. Introducción al motor a pasos	10
2.1.3. Principales características del motor a pasos	10
2.1.4. Partes de un motor a pasos	11
2.1.5. Ventajas e inconvenientes del motor a pasos	13
2.1.6. Características funcionales del motor a pasos	14
2.1.7. Clasificación de los motores a pasos según su excitación	22
2.1.8. Clasificación de los motores a pasos según su construcción	24
2.2. El filtro de microondas	31
2.2.1. Definición del filtro de microondas	31
2.2.2. Matriz S	31
2.2.3. Matriz S del filtro de microondas	33
2.2.4. Ancho de banda	34
2.2.5. Frecuencia central	35
2.2.6. Resonador utilizado en el proyecto	35
3. Instrumentación	39
3.1. El analizador de redes	39
3.1.1. Diferencias entre un analizador vectorial de redes y un analizador de espectros	39
3.1.2. Estructura básica de un analizador de redes	40

3.1.3.	Tipos de medidas	44
3.1.4.	Errores en la medida	45
3.1.5.	Técnicas de corrección de errores y calibrado	45
4.	Implementación del proyecto	49
4.1.	Inicios del proyecto	49
4.2.	Hardware	51
4.2.1.	Funcionalidad del hardware y diagrama de bloques	52
4.2.2.	Entradas y salidas de tensión	56
4.2.3.	Fuente de 5V DC	57
4.2.4.	Fuente de Vreg DC, MCLR e ICSP	58
4.2.5.	Comunicación RS-232	60
4.2.6.	Microchip PIC 16F727	61
4.2.7.	Etapa lógica	63
4.2.8.	Optoacopladores y transistores de potencia	65
4.2.9.	Elaboración física del hardware	67
4.3.	Software	72
4.3.1.	Descripción funcional del software	72
4.3.2.	Programa principal: <i>main_sintonizacion</i>	73
4.3.3.	Configuración del puerto serie: <i>inicializar_puerto</i>	76
4.3.4.	Petición de datos al usuario: <i>peticion_datos</i>	77
4.3.5.	Parametrización del filtro: <i>inicializar_filtro</i>	78
4.3.6.	Cálculo del parámetro S_{21} : <i>calcular_S</i>	78
4.3.7.	Cálculo de la frecuencia central del resonador: <i>calcular_fc</i>	79
4.3.8.	Cálculo de la frecuencia central del resonador en modo simulación: <i>calcular_fc_simulacion</i>	79
4.3.9.	Enviar órdenes al microcontrolador: <i>envia_orden</i>	81
4.3.10.	Finalizar la comunicación serie: <i>finalizar_puerto</i>	82
4.4.	Firmware	84
4.4.1.	Entorno de programación: <i>MPLAB IDE v8.36</i> y <i>MPLAB ICD 2</i>	84
4.4.2.	Características del firmware	86
4.4.3.	Hilo principal del firmware: <i>main.c</i>	88
4.4.4.	Configuración de los puertos del microcontrolador: <i>PORTS.c</i>	89
4.4.5.	Header de los puertos: <i>PORTS.h</i>	90
4.4.6.	Configuración de los registros: <i>REGISTERS.c</i>	90
4.4.7.	Header de los registros: <i>REGISTERS.h</i>	91
4.4.8.	Activar las fases del motor: <i>STATES.c</i>	91
4.4.9.	Header de los estados: <i>STATES.h</i>	93
4.4.10.	Funciones de retardo: <i>DELAY.c</i>	93
4.4.11.	Header de los retardos: <i>DELAY.h</i>	94
4.4.12.	Mapeado de memoria de los registros del microcontrolador: <i>pic16f72x.h</i>	94
4.5.	Estructura de soporte y mecanizado	95
4.5.1.	Estructura de soporte	96
4.5.2.	Puntera del motor	97
4.5.3.	Tornillo de sintonización	98
5.	Resultados experimentales	99
5.1.	Parametrización	100
5.1.1.	Procedimiento y descripción de la parametrización	100

5.1.2.	Resultados de la parametrización	101
5.1.3.	Análisis de los resultados	103
5.1.4.	Conclusiones de la parametrización	104
5.2.	Sintonización descendente de frecuencia	106
5.2.1.	Procedimiento y descripción de la sintonización descendente	106
5.2.2.	Resultados de la sintonización descendente	106
5.2.3.	Análisis de los resultados	107
5.2.4.	Conclusiones de la sintonización descendente	108
5.3.	Sintonización ascendente de frecuencia	109
5.3.1.	Procedimiento y descripción de la sintonización ascendente	109
5.3.2.	Resultados de la sintonización ascendente	109
5.3.3.	Análisis de los resultados	110
5.3.4.	Conclusiones de la sintonización ascendente	111
6.	Conclusiones y líneas de investigación futuras.....	113
6.1.	Conclusiones generales	113
6.2.	Limitaciones del proyecto.....	115
6.3.	Relación entre el ángulo de paso y la respuesta frecuencial del resonador	116
6.4.	Líneas de investigación futuras.....	117
Bibliografía y recursos web.....		119
Bibliografía		119
Páginas web consultadas		121
Anexo A.: Ecuación de la gráfica ‘Par estático normalizado’		123
Anexo B.: Código del Software de Control		125
Archivo <i>main_sintonizacion.m</i>		126
Archivo <i>inicializar_puerto.m</i>		134
Archivo <i>peticion_datos.m</i>		135
Archivo <i>inicializar_filtro.m</i>		137
Archivo <i>calcular_S.m</i>		140
Archivo <i>enviar.cpp</i>		141
Archivo <i>rebre.cpp</i>		145
Archivo <i>calcular_fc.m</i>		150
Archivo <i>calcular_fc_simulacion.m</i>		151
Archivo <i>envia_orden.m</i>		152
Archivo <i>finalizar_puerto.m</i>		155
Anexo C.: Código del Firmware		157
Archivo <i>main.c</i>		158

Archivo <i>PORTS.c</i>	160
Archivo <i>PORTS.h</i>	161
Archivo <i>REGISTERS.c</i>	162
Archivo <i>REGISTERS.h</i>	163
Archivo <i>STATES.c</i>	164
Archivo <i>STATES.h</i>	168
Archivo <i>DELAY.c</i>	169
Archivo <i>DELAY.h</i>	170
Archivo <i>pic16f72x.h</i>	171
Anexo D.: <i>BOM</i>	175
Anexo E.: <i>Layouts y disposición de los componentes en el hardware</i>	177
Anexo F.: <i>Data Sheets</i>	179
Anexo G.: <i>Stepper Motor 55SPM25D7ZA Data Sheet</i>	181

Lista de acrónimos

AC	Alternating Current.	IF	Intermediate Frequency.
ADC	Analog to Digital Converter.	ISR	Interrupt Service Routine.
AR	Analizador de redes.	LO	Local Oscillator.
CLK	Clock.	Matriz S	Matriz de Scattering o pérdidas.
CNC	Computer Numerical Controlled.	MCL	Master Clear.
CPU	Central Processing Unit.	MIPS	Million Instructions per Second.
CTR	Current Transfer Ratio.	PCB	Printed Circuit Board.
DAT	Data.	PIC	Peripheral Interface Controller.
DC	Direct Current.	RAM	Random Access Memory.
DIP	Dual In-line Packages.	RF	Radio Frequency.
DSP	Digital Signal Processor.	SPI	Serial Peripheral Interface Bus.
DUT	Device Under Test.	SPP	Sequenced Packed Protocol.
GND	Ground.	USART	Universal Synchronous/Asynchronous Receiver/Transmitter.
GPIB	General Purpose Interface Bus.	VCO	Voltage Controlled Oscillator.
I²C	Inter-Integrated Circuit.		
ICD	In-Circuit Debugger.		

Índice de figuras

Fig. 1: Diagrama de las conexiones entre los elementos del proyecto.	3
Fig. 2: Fuerza magnética de Lorentz sobre una carga negativa.	7
Fig. 3: Fuerza magnética de Lorentz sobre un conductor de corriente.	8
Fig. 4: Fuerza magnética de Lorentz sobre una espira rectangular de corriente.	8
Fig. 5: Escobillas (en gris) y contactos de los diferentes devanados del rotor de un motor DC.	9
Fig. 6: Esquema del corte transversal de un motor a pasos.	10
Fig. 7: Esquema representativo del estator y el rotor de un motor a pasos.	11
Fig. 8: Motor a pasos con un estator de 4 devanados, 4 polos y 3 dientes por devanado. El rotor es híbrido con polarización Norte y Sur de 15 dientes cada uno.	12
Fig. 9: Motor a pasos con un estator de 8 devanados y 8 polos sin dientes. El rotor tiene 6 dientes con una única polarización.	12
Fig. 10: Gráfico representativo del par estático de un motor a pasos.	15
Fig. 11: Gráfico del par estático normalizado de un motor a pasos con 3 fases y la fase 1 como referencia o paso cero.	18
Fig. 12: Gráfico del par dinámico del motor a pasos 55SPM25D utilizado en este proyecto.	20
Fig. 13: Gráfico del par dinámico y par dinámico de arranque del motor a pasos M35SP-8.	21
Fig. 14: Motor unipolar con dos devanados divididos por la mitad. A la izq. 6 conexiones con su color habitual de cableado y a la dcha. se han unido los puntos medios de los dos devanados para obtener 5 conexiones.	22
Fig. 15: Motor unipolar trabajando como bipolar.	23
Fig. 16: Motor bipolar de dos devanados.	24
Fig. 17: Interruptores necesarios para activar un devanado en los dos sentidos de giro.	24
Fig. 18: Sección de un motor a pasos de reluctancia variable.	25
Fig. 19: Motor con estator magnetizado permanentemente.	26
Fig. 20: Motor con rotor permanentemente magnetizado.	27
Fig. 21: Representación simplificada del modo de trabajo de un motor con rotor permanentemente magnetizado de 7.5° por paso.	27
Fig. 22: Detalle del estator de un motor con el rotor permanentemente magnetizado.	28
Fig. 23: Esquema representativo de un motor a pasos híbrido.	29
Fig. 24: Red bipuerto.	33
Fig. 25: Ancho de banda de un filtro paso banda.	34
Fig. 26: Vista interior del resonador débilmente acoplado utilizado.	36
Fig. 27: Pérdidas de inserción del resonador débilmente acoplado.	36
Fig. 28: Distribución superficial de corriente en el resonador de cuarto de onda.	36
Fig. 29: Modificaciones realizadas sobre el resonador original.	37
Fig. 30: Pérdidas de inserción del resonador modificado a 7,125GHz.	37
Fig. 31: Esquema funcional de los diodos detectores junto con su ancho de banda de trabajo.	42
Fig. 32: Esquema funcional de los receptores sintonizables junto con su ancho de banda de trabajo.	43
Fig. 33: Modelo de corrección de errores mediante la técnica de calibración de un puerto [19].	46
Fig. 34: Modelo de corrección de errores mediante la técnica de calibración de dos puertos [19].	47
Fig. 35: Protoboard del hardware del proyecto.	51
Fig. 36: Inicio del programa Altium Designer 6.	52
Fig. 37: Archivo TOP_PCB.SchDoc. Diagrama e interconexión de los bloques componentes del hardware.	53
Fig. 38: Jerarquía de los directorios del proyecto.	54
Fig. 39: Archivo PCB_PIC.PcbDoc. Vista general del hardware con los dispositivos y las pistas.	55
Fig. 40: Esquema eléctrico de las entradas y salidas de tensión.	56
Fig. 41: Esquema eléctrico de la fuente de 5V DC.	57

Fig. 42: Esquema eléctrico del regulador de tensión, tensión de reset y circuito auxiliar para la conexión del microcontrolador al MPLAB ICD 2.	58
Fig. 43: Esquema eléctrico del bloque RS-232 en el hardware.	60
Fig. 44: Esquema eléctrico del microcontrolador con los leds de control y los pins de expansión.	61
Fig. 45: Esquema eléctrico de la etapa lógica.	64
Fig. 46: Equivalencia en lógica NAND de las puertas estándares.	65
Fig. 47: Esquema eléctrico del aislamiento con optoacopladores, transistores de potencia, diodos de recuperación rápida y conectores del motor y los interruptores.	66
Fig. 48: Layout de la cara superior del PCB.	68
Fig. 49: Layout de la cara inferior del PCB con efecto espejo.	68
Fig. 50: Vista de la cara superior del PCB.	69
Fig. 51: Vista de la cara inferior del PCB.	69
Fig. 52: Imagen del hardware finalizado.	71
Fig. 53: Representación del proceso de sintonización de una frecuencia f_c con $n+3$ pasos.	74
Fig. 54: Gráficas del módulo de los parámetros S_{11} y S_{21} a 8,5GHz del resonador sintonizable utilizado.	75
Fig. 55: Frecuencia de resonancia en cada paso, incremento de frecuencia de resonancia entre pasos consecutivos y pérdidas de inserción en cada paso de una sintonización del resonador utilizado.	75
Fig. 56: Pantalla de inicio del programa MPLAB IDE.	84
Fig. 57: ICSP mediante MPLAB ICD 2.	85
Fig. 58: MPLAB ICD 2 con conexión USB al PC y RJ-12 al hardware	85
Fig. 59: Estructura jerárquica de los documentos del firmware.	86
Fig. 60: Diagrama de flujo de las principales instrucciones del firmware.	89
Fig. 61: Ventana con la configuración de los registros configurables desde el MPLAB IDE.	91
Fig. 62: Diagrama de estados de las fases activas del motor.	92
Fig. 63: Estructura de soporte con el hardware y el motor.	95
Fig. 64: Estructura de soporte del motor a pasos y el hardware.	96
Fig. 65: Detalle de la puntera multiusos encajada en el eje del rotor.	97
Fig. 66: Tornillo de sintonización tipo Allen métrico 5mm.	98
Fig. 67: Resultados de la parametrización del resonador con un rango de frecuencias sintonizables de 5,9GHz a 8,9GHz, y un rango de medida del AR de 5GHz a 10GHz con 200 puntos.	101
Fig. 68: Pérdidas de inserción en el paso 2 con $f_c=5,9625\text{GHz}$.	102
Fig. 69: Pérdidas de inserción en el paso 5 con $f_c=6,5\text{GHz}$.	102
Fig. 70: Pérdidas de inserción en el paso 10 con $f_c=7,125\text{GHz}$.	102
Fig. 71: Pérdidas de inserción en el paso 15 con $f_c=7,5375\text{GHz}$.	102
Fig. 72: Pérdidas de inserción en el paso 26 con $f_c=8,1125\text{GHz}$.	102
Fig. 73: Pérdidas de inserción en el paso 62 con $f_c=8,7375\text{GHz}$.	102
Fig. 74: Resultados de la sintonización descendente del resonador a 7GHz, con un rango de medida del AR de 5GHz a 10GHz y 200 puntos.	106
Fig. 75: Pérdidas de retorno (S_{11}) y pérdidas de inserción (S_{21}) del resonador a la frecuencia sintonizada de 7GHz.	107
Fig. 76: Resultados de la sintonización ascendente del resonador a 8,5GHz, con un rango de medida del AR de 5GHz a 10GHz y 200 puntos.	109
Fig. 77: Pérdidas de retorno (S_{11}) y pérdidas de inserción (S_{21}) del resonador a la frecuencia sintonizada de 8,5GHz.	110
Fig. 78: Diagrama general del hardware con los dispositivos y las pistas (en rojo: superior, en azul: inferior)	177

Índice de tablas

Tabla 1: Tensiones de reset del microcontrolador. _____	59
Tabla 2: Tensión de alimentación del microcontrolador. _____	60
Tabla 3: Entradas al microcontrolador. _____	62
Tabla 4: Salidas del microcontrolador. _____	62
Tabla 5: Ecuaciones lógicas de las fases del motor. _____	64
Tabla 6: Tabla de verdad de las fases del motor. _____	64
Tabla 7: Configuración del puerto serie. _____	76
Tabla 8: Codificación de la instrucción de iniciar la comunicación con el microcontrolador. _____	77
Tabla 9: Codificación de la instrucción de comunicación iniciada correctamente por el microcontrolador. _____	77
Tabla 10: Incremento del puntero del vector de frecuencias fcv en modo simulación. _____	80
Tabla 11: Distribución de los 8 bits de la palabra enviada por el puerto serie. _____	81
Tabla 12: Codificación de las instrucciones de giro que envía el PC hacia el microcontrolador. _____	81
Tabla 13: Codificación de las respuestas a las órdenes de giro que envía el microcontrolador hacia el PC. _____	82
Tabla 14: Codificación de la instrucción de finalizar la comunicación con el microcontrolador. _____	82
Tabla 15: Codificación de la instrucción de comunicación finalizada correctamente por el microcontrolador. _____	83
Tabla 16: Características físicas de la estructura de soporte. _____	97
Tabla 17: BOM (Bill Of Materials) del hardware implementado. _____	175
Tabla 18: Links de los Data sheet de los integrados utilizados. _____	179
Tabla 19: Links de los Data Sheet de los instrumentos de medida. _____	179

1. Introducción

1.1. Introducción

En comunicaciones y especialmente en radiocomunicaciones es habitual tener que insertar en los sistemas elementos como filtros, amplificadores, acopladores o resonadores para adecuar la señal a las necesidades que se requieran según su aplicación. De entre todos estos elementos, el objeto de estudio de este proyecto es un resonador de microondas en el rango de 5GHz a 10GHz.

En el tipo de resonador elegido se puede ajustar la frecuencia central mediante un tornillo situado en la parte superior que, dependiendo de la profundidad de penetración que se aplique, desplaza la frecuencia de resonancia arriba o abajo.

Como veremos más adelante, el proyecto se centra en el diseño de un software de control, un hardware con su respectivo firmware y una estructura de soporte para un motor a pasos que actúa sobre el tornillo del resonador y que permite ajustar automáticamente la frecuencia de resonancia del mismo. En resumidas cuentas, el usuario introduce los parámetros necesarios en una aplicación Matlab y tras pocos segundos el motor habrá sintonizado el resonador a la frecuencia deseada automáticamente.

Aunque para la gente que trabaja habitualmente con este tipo de dispositivos es completamente normal tener que dedicar cierto tiempo al ajuste de frecuencias, no deja de ser un trabajo repetitivo y que no tiene secreto alguno. Al igual que el resonador, hay muchos dispositivos de radiofrecuencia que son ajustables o sintonizables mecánicamente mediante tornillos y que, por lo tanto, puede aplicárseles éste proyecto para automatizar el ajuste de parámetros que requiere la aplicación de un par mecánico para ajustar la profundidad de penetración de un tornillo.

1.2. Motivación y contexto del proyecto

La demanda de componentes de microondas inteligentes, capaces de ajustar su frecuencia de operación automáticamente es cada vez mayor. Ejemplos de aplicaciones son los filtros reconfigurables capaces de ajustar sus parámetros para diversas aplicaciones móviles terrestres o satelitales. Los filtros reconfigurables pueden adaptarse a pequeños desajustes en frecuencia central causados por cambios climáticos. Es por eso que la complejidad de los componentes que operan en múltiples bandas hace que cada vez más los ingenieros tengan que diseñar nuevos dispositivos que sean capaces de mejorar la calidad de sus antecesores y agilizar su ajuste.

Selectividad y ancho de banda son cada vez más estrechos y esto hace que el ajuste de los mismos se convierta en una parte importante a tener en cuenta en todo sistema de transmisión. La versatilidad que puede ofrecer un dispositivo que pueda auto-ajustar sus características podría ser de gran interés sobretodo en el ámbito de la investigación, en donde los re-ajustes son constantes debido fundamentalmente a los cambios de la temperatura y las diferentes tolerancias de los componentes.

Es por esto que el proyecto nace de la necesidad de poder automatizar de manera eficiente y precisa la sintonización de resonadores, filtros y otros dispositivos que trabajen en la banda de microondas y que precisen de un ajuste usando tornillos.

No debemos olvidar que muchas veces estos elementos pueden llegar a tener doce, quince o hasta veinte tornillos destinados a ajustar mecánicamente diferentes parámetros. En estos casos es de gran ayuda algún sistema que permita automatizar la sintonización de los tornillos.

Una aplicación de este proyecto y relacionado con lo anteriormente dicho, es que podría programarse una rutina de ajuste de los diferentes parámetros en el programa de control de tal manera que, estableciendo previamente el orden de los tornillos e introduciendo el valor de los parámetros que se desean conseguir, se realice una primera pasada por todos los tornillos para ajustar sin mucha precisión las características básicas y ya en una segunda realizar el sintonizado fino dedicándole más tiempo a cada parámetro.

Incluso se podría dotar al programa de control de una cierta “autonomía” y que nos guiara para situar el motor en los diferentes tornillos y re-ajustar el parámetro en cuestión. Evidentemente esta opción requiere de un conocimiento muy detallado del dispositivo a ajustar y la diferenciación clara de las funciones de cada tornillo.

Es por todo lo expuesto anteriormente que creo en la firme decisión de realizar este proyecto y que gracias a los resultados obtenidos pueda contribuir con mi pequeño granito de arena al avance implacable de la ciencia y la tecnología.

1.3. Objetivo del proyecto

El objetivo del proyecto es diseñar un sistema que permita el ajuste automático de la frecuencia de resonancia de un resonador de microondas mediante un motor a pasos.

La sintonización de la frecuencia de resonancia se realiza mediante un tornillo de sintonización, y para actuar sobre él se ha escogido un motor a pasos con una puntera intercambiable para adaptarse a las distintas cabezas de los tornillos. Para hacer girar el motor se ha diseñado un hardware específico controlado por un microcontrolador que a su vez se comunica con el software de control y recibe las instrucciones para girar el motor en un sentido u otro. A su vez, el resonador de microondas está conectado a un analizador de redes para medir su matriz S y poder obtener su respuesta frecuencial. El analizador de redes envía los datos al software de control y éste calcula la frecuencia central a la que está sintonizado el resonador decidiendo subir o bajar de frecuencia para centrarlo a la frecuencia que desea el usuario.

Así, en función de la penetración del tornillo en la cavidad del resonador se puede modificar la frecuencia central. De esta forma, si queremos que el resonador se centre en una frecuencia superior, sólo tenemos que destornillar el tornillo de sintonización.

En la siguiente figura se muestra un esquema de las conexiones entre los distintos elementos.

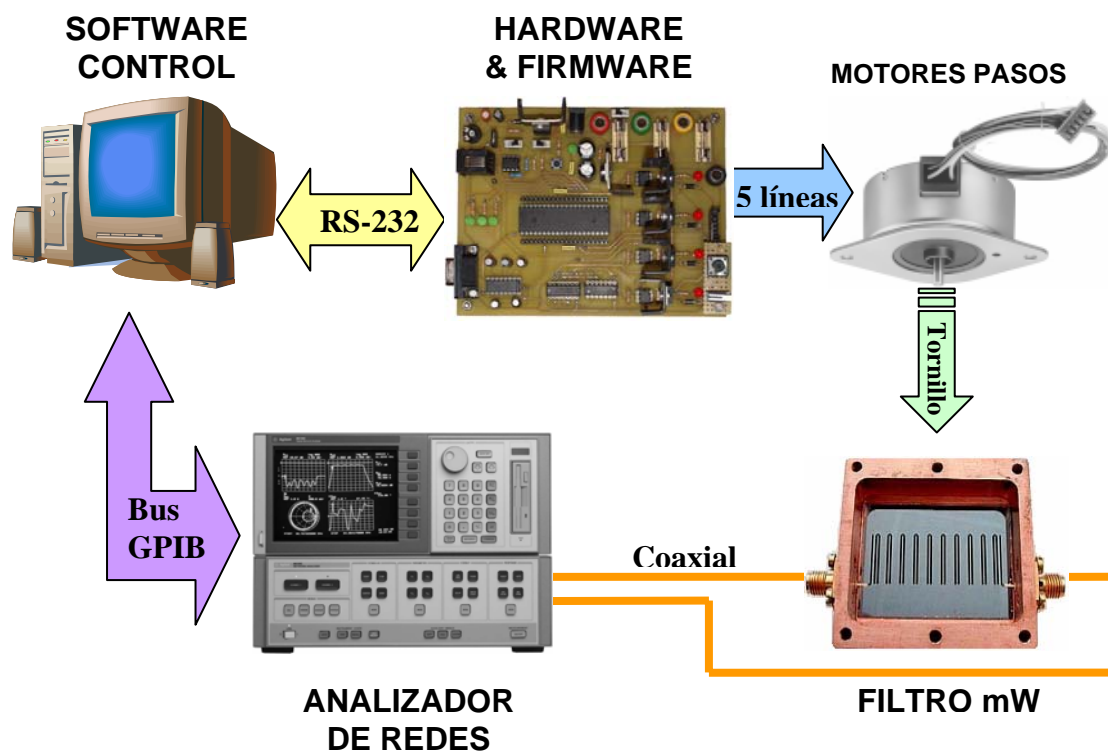


Fig. 1: Diagrama de las conexiones entre los elementos del proyecto.

El proyecto se divide en 4 tareas principales a completar:

- **Diseño de la aplicación de control (software de control):** se trata de un algoritmo programado en *Matlab* que analiza y procesa las medidas provenientes del analizador de redes y calcula la frecuencia de resonancia a la que está centrado el resonador. Una vez calculada la frecuencia de resonancia, envía las órdenes al hardware para hacer girar el motor en un sentido u otro variando así la frecuencia de resonancia y sintonizarla a la frecuencia introducida por el usuario.
- **Diseño del *interface* (firmware) y el *driver* (hardware):** son los encargados de comunicarse e interpretar las órdenes provenientes del software de control y entregar la corriente necesaria a las fases del motor respectivamente.
- **El motor a pasos y la estructura de soporte:** se analiza el motor escogido, se optimiza el diseño y se acopla el eje del motor al tornillo del resonador mediante una puntera estándar con puntas intercambiables para mejorar la versatilidad del conjunto. También se diseña un soporte para que el motor pueda situarse encima del resonador y pueda ubicarse el hardware diseñado al lado.
- **Presentación de los resultados:** se implementa un entorno que combina líneas de comando y una parte gráfica dentro de *Matlab* para que la interacción con el usuario sea más cómoda y agradable, a la par que se presentan los resultados de forma comprensible y sencilla.

Llegados a este punto, podemos describir la secuencia lógica que seguirá el programa. Primero se introducen los datos necesarios en el software de control, entre ellos la frecuencia a la que se quiere centrar el resonador, el rango y número de puntos de la medida, la dirección GPIB del analizador de redes, etc. Una vez introducidos estos datos, empieza el autoajuste.

El analizador de redes mide la matriz S del resonador y se la envía al software de control para que calcule la frecuencia de resonancia y decida si es la frecuencia a la que el usuario desea centrar el resonador. En caso contrario decide que el motor gire en un sentido u otro dependiendo de si debe aumentar o disminuir la frecuencia. Este proceso de medida, cálculo y decisión se repite hasta obtener el ajuste a la frecuencia deseada con el mínimo error posible dadas las características del motor, el analizador de redes y el procedimiento utilizado.

Una vez conseguidos estos objetivos, se analizarán los resultados obtenidos y se propondrán futuras líneas de investigación y mejora del proyecto junto con otras aplicaciones que se le podría dar.

1.4. Organización de la memoria

La memoria del proyecto se organiza en seis capítulos más los anexos. El primero es la introducción en donde se describe el escenario del proyecto y se ayuda al lector a familiarizarse con la temática de los capítulos posteriores. En este primer capítulo encontramos una breve descripción del proyecto, la motivación para realizarlo y los objetivos a alcanzar.

En el segundo capítulo se explican los fundamentos teóricos que atañen a los componentes del proyecto. En este capítulo se describe el comportamiento del motor a pasos y sus principales características, así como los fundamentos físicos de los filtros de microondas y sus características principales.

El tercer capítulo está destinado a la instrumentación utilizada. En él se describen las características de los elementos de medición utilizados, la metodología seguida para hacer las mediciones, su calibrado y su contribución al error final.

En el cuarto capítulo se describe la implementación de todo el proyecto, los pasos seguidos junto con su argumentación y los problemas encontrados junto con su solución. Este apartado está dividido en cuatro grandes grupos: Hardware, Software, Firmware y Estructura de soporte que son los cuatro grandes apartados en los que se ha dividido el proyecto.

El capítulo quinto aborda los resultados experimentales, en donde se describen los resultados obtenidos así como un análisis de los mismos y la interpretación de las gráficas obtenidas.

El último capítulo corresponde a las conclusiones y líneas de investigación futuras. Aquí se exponen las conclusiones que se extraen del diseño y el análisis del sistema y se deja la puerta abierta a nuevos proyectos que, aprovechando este mismo, pueden ampliar su aplicación a dispositivos más complejos.

Para finalizar encontramos las referencias bibliográficas utilizadas para la realización de este proyecto y los anexos, que contienen información relevante para el proyecto pero que no se incluyen dentro de ninguno de los apartados principales. En el Anexo A se encuentra el código de la *Fig. 11* a partir de la extrapolación de una ecuación extraída de la referencia bibliográfica [1]. En el Anexo B se encuentra el código de las funciones componentes del software de control programado en *Matlab*. En el Anexo C se hallan los archivos componentes del firmware programados en C dentro de un entorno *MPLAB*. El Anexo D es el BOM (*Bill Of Materials*) de la placa hardware diseñada y el Anexo E contiene un esquema con la disposición de los componentes del hardware y sus pistas. En el Anexo F se encuentran los links a las páginas web de los fabricantes de los integrados utilizados en el hardware con sus *Data Sheets* correspondientes. Y el Anexo G es el *Data Sheet* del motor a pasos utilizado.

2. Fundamentos teóricos

2.1. El motor a pasos

Antes de empezar con las características de los motores a pasos, se hace una introducción de los motores de corriente continua y sus fundamentos físicos.

2.1.1. El motor eléctrico de corriente continua

Un motor es un actuador que transforma energía eléctrica en energía mecánica, es decir, en movimiento. En los motores podemos distinguir dos partes claramente diferenciadas: el rotor y el estator. El estator está compuesto principalmente de un imán que es el encargado de generar el campo magnético. En el rotor encontramos diferentes bobinados de cobre por los que circula la corriente eléctrica.

Debido a la fuerza de Lorentz, cuando una carga se mueve dentro de una región afectada por un campo electromagnético, ésta experimenta una fuerza tal que:

$$\vec{F} = q \cdot (\vec{E} + \vec{v} \times \vec{B}) \quad (2.1)$$

En donde:

\vec{F} es la fuerza que experimenta la carga.

q es la carga que se mueve.

\vec{E} es el campo eléctrico en la región.

\vec{v} es la velocidad de la carga.

\vec{B} es el campo magnético en la región.

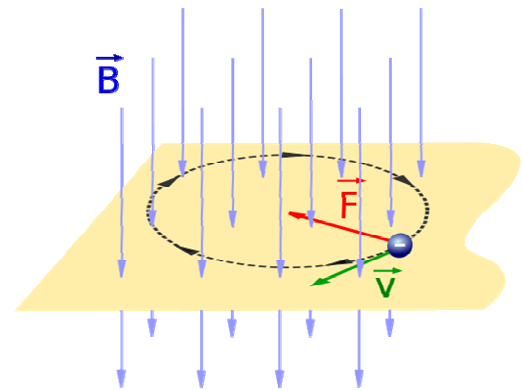


Fig. 2: Fuerza magnética de Lorentz sobre una carga negativa.

Debido al producto vectorial de la velocidad y el campo magnético, la carga sólo experimentará fuerza debido al campo magnético cuando la dirección de la carga y las líneas de campo magnético no sean paralelas. Y en consecuencia la fuerza será máxima cuando sean perpendiculares.

Si entendemos que una carga en movimiento experimenta una fuerza cuando se mueve dentro de un campo magnético, no resulta difícil extrapolar este concepto y extenderlo a una línea de corriente, ya que no deja de ser un conjunto de cargas moviéndose a lo largo de un conductor. Aplicando diferentes principios de física se puede demostrar que la fuerza experimentada por una línea de corriente dentro de un campo magnético es:

$$\vec{F} = \int_L I \cdot d\vec{l} \times \vec{B} \quad (2.2)$$

En donde:

\vec{F} es la fuerza que experimenta la línea de corriente.

L es la longitud de la línea afectada por el campo magnético.

I es el módulo de la corriente que circula por la línea.

$d\vec{l}$ es el diferencial de longitud de la línea.

\vec{B} es el campo magnético en la región.

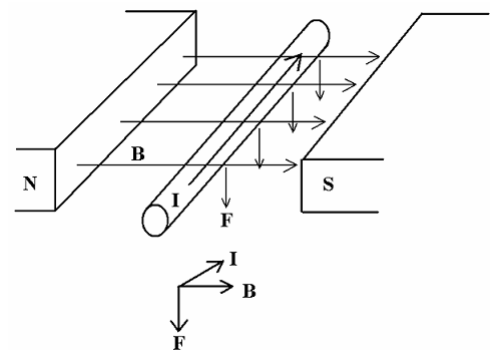


Fig. 3: Fuerza magnética de Lorentz sobre un conductor de corriente.

Se acaba de demostrar que la circulación de cargas eléctricas por un conductor que se encuentra en una región afectada por un campo magnético, crea una fuerza sobre el mismo conductor y esta fuerza sigue la dirección resultante del producto vectorial entre la corriente y el campo magnético.

Ahora bien, si en lugar de una línea de corriente tenemos una espira rectangular dentro de la región magnética, es fácil comprender que sobre cada uno de los cuatro costados de la espira actuará una fuerza diferente ya que la dirección de las cargas es distinta en cada uno de ellos.

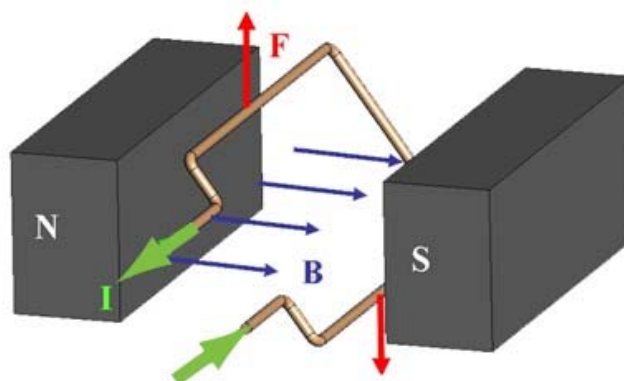


Fig. 4: Fuerza magnética de Lorentz sobre una espira rectangular de corriente.

Como se ha mencionado anteriormente, si la dirección de las cargas y el campo magnético son paralelos, la fuerza en el conductor es nula. Por lo tanto, hay un punto en el que sobre los costados paralelos al eje N-S del campo magnético (costados cortos) no actúa ninguna fuerza. Y conforme la espira va girando, la fuerza que actúa en estos mismos costados se cancela debido a su signo contrario.

A diferencia de los costados anteriores, en los otros dos costados perpendiculares al eje N-S del campo magnético (costados largos), se crea un par de fuerzas que hace girar la espira. Esto es debido al sentido contrario de la corriente (vector velocidad de las cargas) ya que cuando la fuerza de un costado tira hacia arriba, la del otro tira hacia abajo.

Pero esto sólo sucede durante 180° . Si no se invierte el sentido de la corriente, el motor se quedará clavado ya que el par desaparece al contraponerse las fuerzas de los dos costados sobre el mismo eje de giro. Entonces es cuando entran en juego las escobillas del motor.

Las escobillas del motor son los puntos por donde se inyecta la corriente a las espiras del rotor. Durante el contacto de la espira con la escobilla, la corriente circula en un mismo sentido mientras que cuando se encuentra en la posición opuesta, la corriente circula en sentido contrario debido a que la entrada y salida de corriente se han intercambiado. De esta forma se consigue un nuevo par en sentido contrario al anterior. Repitiendo este fenómeno conseguimos que las espiras giren sin parar mientras circule corriente eléctrica por ellas.

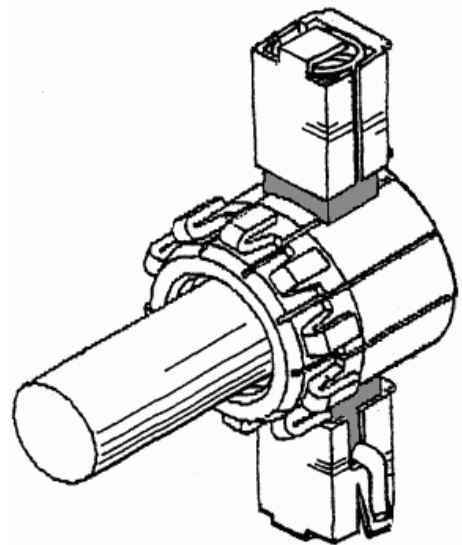


Fig. 5: Escobillas (en gris) y contactos de los diferentes devanados del rotor de un motor DC.

Una vez entendido el principio de funcionamiento y aplicando superposición, si en lugar de tener una espira tenemos N espiras formando un único devanado, la fuerza se multiplicará por N si en cada espira circula la misma corriente. De esta forma tendremos más par de giro pero mayor consumo energético.

Y si en lugar de un único devanado tenemos M devanados posicionados en diferentes ángulos, podemos asegurar que en una vuelta del rotor tendremos M posiciones de par máximo y nunca se clavarán el rotor en una posición de equilibrio de fuerzas.

2.1.2. Introducción al motor a pasos

El motor a pasos es un tipo de motor eléctrico que excita sus bobinados con un cierto desfase mediante señales cuadradas. Esta característica facilita mucho la interfase de los motores y especialmente hoy en día que la mayoría de sistemas electrónicos trabajan bajo control de microprocesadores que utilizan bits para comunicarse.

Aunque se han estado usando los motores a pasos desde hace más de medio siglo, éstos han retomado su importancia a partir del nacimiento de los microcontroladores y gracias a que su eficacia y prestaciones han aumentando debido a su eficaz e innovador sistema de fabricación pudiendo reducir así sus costes.

De todas formas, los motores a pasos son inherentemente menos eficientes en la conversión de energía electromagnética en comparación con los motores de corriente continua o alterna, pero tienen otras propiedades que los hacen ser indispensables en muchas aplicaciones industriales y de consumo de hoy en día. Se puede destacar su uso en periféricos de ordenador, impresoras, relojes, sistemas de seguimiento del sol en paneles fotovoltaicos, posicionamiento de antenas, equipamiento fotográfico y óptico, equipos médicos, robots,...

Y no solamente pueden ser utilizados como mecanismos de transformación de la energía eléctrica en movimiento, sino que también pueden utilizarse como convertidores digital-analógico, contadores de pulsos, etc.

2.1.3. Principales características del motor a pasos

Un motor a pasos es un motor de corriente continua en el que se ha invertido la funcionalidad del estator y del rotor. En un motor de corriente continua convencional el estator se encarga de crear un campo magnético constante en tiempo y en espacio. El rotor, por su parte, es el encargado de crear el par de fuerzas que lo hace girar gracias a la corriente que circula por sus devanados y que los polariza, haciendo que se oriente según el campo magnético creado por el estator.

Sin embargo, en el motor a pasos, el campo magnético (permanente o no) se encuentra en el rotor y el estator hace que se oriente según la corriente que circula por sus devanados. La corriente polariza un núcleo de hierro dulce y éste se imanta creando un polo norte y sur según el sentido de la corriente. Entonces el rotor se ve obligado a orientarse según el campo generado en el estator.

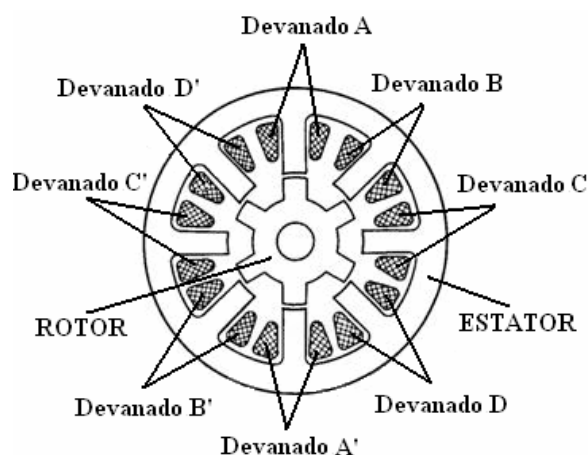


Fig. 6: Esquema del corte transversal de un motor a pasos.

Dependiendo del tipo de motor a pasos, si no se modifica la corriente de los devanados del estator, el motor permanece clavado en esa posición. Si por el contrario cambiamos la corriente, el motor realizará un paso a la derecha o izquierda según se active el devanado de la derecha o de la izquierda.

2.1.4. Partes de un motor a pasos

Aunque existen variaciones en función del tipo específico de motor, hay ciertas partes que son comunes en todos los motores a pasos, debido a que el principio de funcionamiento es igual en todos ellos. Así, podemos destacar las siguientes partes:

- **Estator**

Es la parte fija y forma la estructura principal alrededor del motor. Tiene forma cilíndrica y está formado habitualmente por varios dientes organizados en grupos de cuatro a diez formando un único polo. Suelen tener devanados independientes que se polarizan cuando circula corriente por los mismos. El campo magnético creado en los devanados polariza los dientes y éstos atraen al polo opuesto del rotor provocando el giro. Los dientes están situados de tal manera que favorecen el giro del rotor al cambiar la polarización de los devanados adyacentes y dependiendo del número de pasos por vuelta del motor encontramos más o menos dientes por polo.

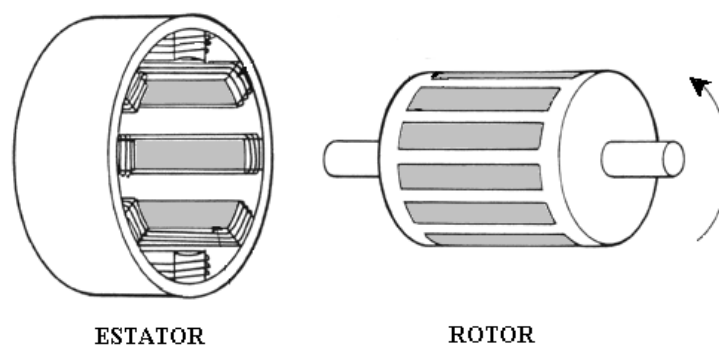


Fig. 7: Esquema representativo del estator y el rotor de un motor a pasos.

- **Rotor**

El rotor es la parte móvil del motor y se trata de una pieza cilíndrica que gira alrededor de su eje longitudinal. Suele estar magnetizado permanentemente y define un número determinado de polos magnéticos, aunque también encontramos rotores que no lo están (como veremos en el apartado 2.1.8 *Clasificación de los motores a pasos según su construcción*). Al igual que el estator, el rotor puede tener dientes que se alinean perfectamente con los del estator. De esta manera, un diente del estator polarizado como norte se alinea con un diente del rotor polarizado como sur y viceversa.

○ Bobinados, devanados y fases

Los bobinados del estator son cada uno de los bloques o rollos de hilo de cobre. No obstante un bobinado puede tener más de un devanado. Entonces si tenemos dos devanados que pertenecen a un mismo bobinado diremos que se trata de un bobinado bifilar, o bobinado simple si solamente tiene uno.

Por fase entendemos cada una de las diferentes disposiciones eléctricas de los devanados. Así, en un motor con dos bobinados y dos devanados por bobina encontramos cuatro u ocho fases dependiendo de si el motor es unipolar o bipolar. Sin embargo es muy habitual comercialmente igualar el número de fases al número de devanados, con lo cual en el caso anterior tendríamos cuatro fases.

○ Polos

Con “polos” nos referimos al número de polos magnéticos que tiene el rotor y/o el estator. Es importante no confundir el número de polos con el número de dientes ya que en general son diferentes y dependen del tipo de motor. A continuación se puede ver un ejemplo muy ilustrativo.

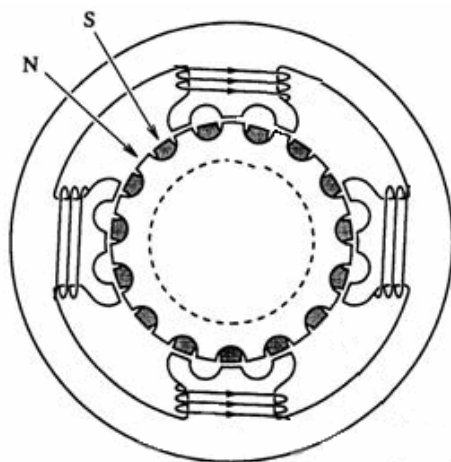


Fig. 8: Motor a pasos con un estator de 4 devanados, 4 polos y 3 dientes por devanado. El rotor es híbrido con polarización Norte y Sur de 15 dientes cada uno.

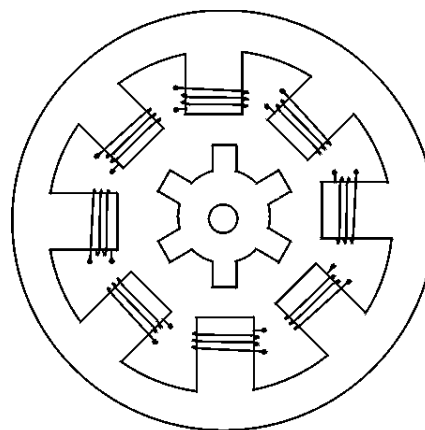


Fig. 9: Motor a pasos con un estator de 8 devanados y 8 polos sin dientes. El rotor tiene 6 dientes con una única polarización.

2.1.5. Ventajas e inconvenientes del motor a pasos

Los motores a pasos se pueden utilizar en sistemas de lazo abierto, o sea, sin realimentación o *feedback*. Esta característica es muy importante porque simplifica en gran medida el sistema, eliminando el grave problema de la estabilidad y se reduce el coste final de diseño y del producto. De todas formas nada impide que un motor a pasos opere en lazo cerrado, y de hecho, es una técnica muy utilizada.

La segunda gran ventaja es la interfase digital que presenta un motor a pasos. Un sistema basado en un microprocesador puede controlar directamente un motor a pasos sin ningún tipo de circuitería adicional a excepción de cuatro transistores que permitan la inyección de potencia a los devanados del motor.

Además, podemos tener acotado el error de posicionamiento ya que el error de posición no es acumulativo. Por otro lado es muy invariable y mayoritariamente se debe a errores mecánicos de fabricación del motor, lo cual implica que se trate de un error repetitivo y no aleatorio. Estas propiedades hacen que el error sea fácil de calibrar.

El motor a pasos ofrece el máximo par cuando está parado (siempre que se inyecte corriente en los devanados). Esto lo hace ser muy útil en aplicaciones cuya estabilidad después del régimen transitorio sea muy importante. Además se pueden conseguir velocidades muy bajas de giro con la carga acoplada sobre el eje del rotor.

La ausencia de escobillas hace del motor a pasos un motor de larga vida útil comparado con otros de DC. Si la electrónica no falla y no se sobrecarga el motor, la vida útil del mismo la marcan los rodamientos de su eje que terminan estropeándose por el desgaste sufrido.

Los inconvenientes de los motores a pasos dependen de la aplicación que se les dé. Pueden conseguir resoluciones muy buenas para la mayoría de aplicaciones pero en cambio puede ser un factor limitante para otras aplicaciones. De hecho, en otros tipos de motores como los de continua, la resolución que se consigue no depende del motor en si mismo sino del codificador de posición que se utilice.

Otra limitación es la potencia que puede ofrecer un motor a pasos. En principio no existe ningún límite físico para construir motores a pasos enormes, pero en la realidad solo existen en unos tamaños limitados. Esto es debido a que en las aplicaciones de gran potencia la eficiencia es un factor primordial a la hora de escoger el motor a utilizar, y la diferencia de coste es muy grande en comparación a un codificador de posición. Por lo tanto se opta por un motor DC de gran eficiencia controlado por un codificador de posición con retroalimentación.

Aunque se ha dicho que el error de posición no es acumulativo y que se puede calibrar sencillamente, el error puede cambiar con el peso de las cargas cuando se utiliza en lazo abierto. También cabe destacar como inconveniente que no siempre tiene una respuesta inicial

suficientemente rápida y el tiempo de giro puede tener valores por encima del deseado en según que aplicaciones.

2.1.6. Características funcionales del motor a pasos

A continuación se describen las características más importantes de los motores a pasos. La primera y más importante es el ángulo de paso ya que nos marca la resolución del motor.

○ Ángulo de paso

El ángulo de paso o, equivalentemente el número de pasos por vuelta, es una de las características fundamentales de los motores a pasos. Depende de la construcción del motor y básicamente de la relación entre el número de polos del estator y el rotor. Actualmente en el mercado podemos encontrar motores desde 90 grados de paso hasta fracciones de grado, pasando por valores estándares o hasta hechos a medida. El valor más usual y el utilizado en este proyecto es el de $7,5^\circ$ o 48 pasos de imán permanente, aunque también es de gran comercialización el de $1,8^\circ$ o 200 pasos por vuelta.

En realidad el ángulo de paso es el ángulo que gira el rotor si activamos las diferentes fases alternativamente, pero se puede conseguir que el rotor avance a pasos más pequeños utilizando técnicas como el medio paso o el micropaso.

○ Par estático (*holding torque*)

El par estático indica la relación que hay entre el desplazamiento del rotor y el par que se aplica sobre el eje del motor cuando el motor está excitado con su tensión nominal. En las gráficas de par estático podemos ver la cantidad de grados que nos hará girar un cierto par aplicado sobre el eje del motor (causado por la carga).

El par estático es, normalmente, más alto que el par dinámico y actúa como freno manteniendo la carga en posición.

En la posición de paso o reposo, los dientes del rotor y del estator están perfectamente alineados y no se produce ningún par. Si el rotor es desplazado fuera de la posición de equilibrio debido a la carga, se desarrolla una fuerza magnética entre los dientes del rotor y del estator, formando un par que obliga al rotor a volver a la posición de paso. Cuando la dirección de desplazamiento del rotor es negativa produce un par positivo, y cuando es positiva se produce un par negativo.

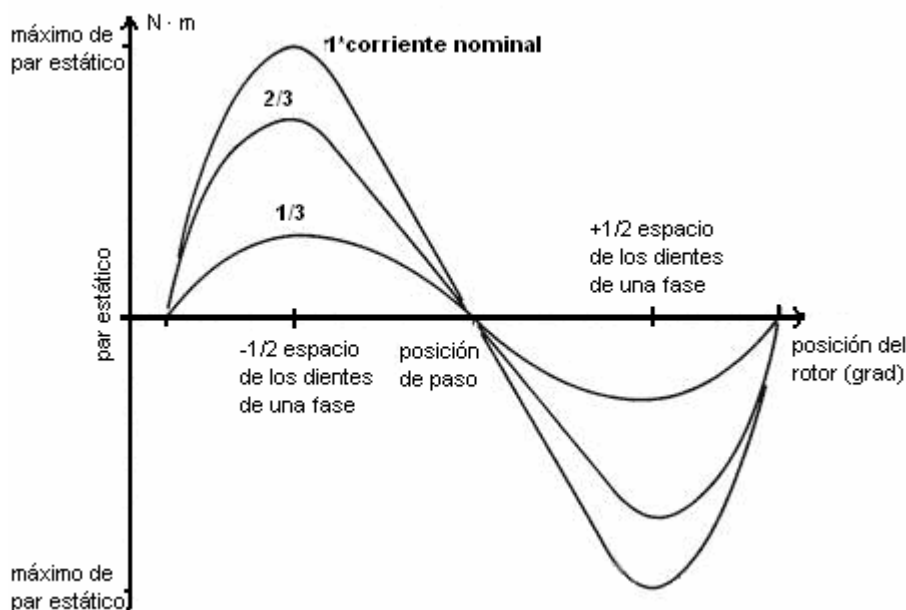


Fig. 10: Gráfico representativo del par estático de un motor a pasos.

En la figura anterior se puede observar una gráfica representativa del par estático. Esta gráfica es teórica y dependiendo de cada motor y sus características la gráfica varía el valor de los máximos y su posición. El par estático tiene un valor nominal siempre positivo, pero debido al sentido en que se aplique la fuerza, se utiliza un convenio de signo positivo o negativo.

Para un mismo motor a pasos, el par estático depende de la corriente inyectada en los devanados y del número de fases activas a la vez. La gráfica del par estático es periódica con periodo $2 \cdot n$ pasos, donde n es el número de fases.

Por ejemplo, en un motor a pasos con 3 fases, la gráfica de par estático tiene un periodo de 6 pasos. Es decir, encontraremos un máximo positivo cada 6 pasos.

Es importante remarcar que existen puntos de equilibrio cada n pasos y se corresponden con la posición de paso (cuando los dientes del rotor y del estator están alineados y no les afecta la carga).

En muchas ocasiones, el fabricante sólo especifica un valor del par estático. Este valor se refiere al máximo de la gráfica. Cuando el par aplicado supera este valor, el motor entra en una zona inestable y el rotor salta hasta la siguiente posición estable.

○ Exactitud del paso

La exactitud del paso mide la diferencia entre la posición teórica del rotor y la real debido a la carga. Depende básicamente de las tolerancias de las diferentes piezas

mecánicas que componen el motor y es habitual que tenga valores inferiores al 5% del paso nominal. De hecho, cada paso tiene un error diferente y cuando hablamos de la exactitud del paso nos referimos al valor medio. El valor que nos dan los fabricantes suele ser válido para cuando el motor se mueve de paso a paso y, habitualmente, con dos fases activas para reducir al máximo el posible error.

Otro factor que hace variar la posición del rotor y aumenta la inexactitud del paso es el hecho de tener un par en el eje del rotor debido a la carga. Este factor no está incluido en la medida que nos proporciona el fabricante, y debemos calcularlo de la siguiente forma dependiendo del par estático que tenga el motor.

La ecuación 2.3 es una aproximación del par estático visto en el apartado anterior (ver apartado *Par estático*). El par estático se asemeja a una forma sinusoidal (ver *Fig. 10* en pág. anterior), y como se observa en la siguiente ecuación, se puede aproximar por un seno alcanzando como máximo el valor máximo del par estático.

$$T = T_0 \cdot \text{sen}\left(\frac{2 \cdot \pi}{2 \cdot n \cdot \theta_p} \cdot \theta\right) \quad (2.3)$$

En donde:

T es el par estático.

T_0 es el par estático máximo.

θ_p es el ángulo de paso del motor.

θ es el ángulo posición del rotor.

n es el número de fases del motor.

Como se puede observar en la ecuación anterior, se tiene el par máximo cuando el argumento del seno sea múltiplo impar de $\pi/2$.

$$\frac{2 \cdot \pi}{2 \cdot n \cdot \theta_p} \cdot \theta = (2 \cdot m - 1) \cdot \frac{\pi}{2} \quad m \in \mathbb{N} \quad (2.4)$$

Operando obtenemos la siguiente condición:

$$\theta = (2 \cdot m - 1) \cdot \frac{(n \cdot \theta_p)}{2} \quad m \in \mathbb{N} \quad (2.5)$$

Así, cuando el ángulo de posición del rotor sea un múltiplo impar de la mitad del ángulo total de todas las fases, el par estático presentará un máximo de valor T_0 , es decir, cuando el rotor esté desplazado en el punto medio de dos dientes activos de la misma fase. Este resultado obtenido es coherente, ya que en ese punto (la mitad de dos dientes activos de la misma fase) es en dónde haríamos el cambio de diente por proximidad, siendo atraídos por el siguiente diente de la misma fase.

El par máximo lo podemos ver como la máxima fuerza que pueden hacer los bobinados del estator sobre el rotor para que el motor funcione correctamente. Si se aplica un par mayor que el máximo en el rotor, el motor saltará a una posición indeterminada o girará indefinidamente hasta que no se deje de aplicar el par.

La ecuación 2.3 es válida si tomamos como referencia la fase a estudiar. Es decir, que podemos extrapolar esta ecuación para todas las fases pero introduciendo un desfase calculado como el ángulo entre la fase referencia y la fase de estudio. Por ejemplo, si tomamos como referencia la fase 1, la ecuación del par estático de la fase k sería:

$$T_{k1} = T_{0k} \cdot \text{sen} \left(\frac{2 \cdot \pi}{2 \cdot n \cdot \theta_p} \cdot \theta + 2 \cdot \pi \cdot \frac{(k-1)}{n} \right) \quad (2.6)$$

En donde:

T_{k1} es el par estático de la fase k tomando como referencia de pasos la fase 1.

T_{0k} es el par estático máximo de la fase k.

θ_p es el ángulo de paso del motor.

θ es el ángulo posición del rotor tomando como referencia angular la fase 1.

n es el número total de fases del motor.

k es la fase a estudiar.

Lo que se quiere expresar con esta ecuación, es que el par estático depende de la fase activa en cada momento y que hay una íntima relación entre el par y el ángulo desplazado del rotor. Así, a más ángulo desplazado, más fuerza se tiene que hacer para mantener el rotor en esa posición hasta que, inevitablemente, se salta al siguiente diente de la misma fase.

También se puede ver el par estático como una fuerza de retención que intenta que el rotor se sitúe en el lugar adecuado para cada paso (posición de paso). Así, lo que intenta evitar el par estático es el desplazamiento involuntario del rotor hacia otras posiciones

cercanas. Es por este motivo que interpretamos el par como positivo o negativo si tomamos como referencia una misma dirección. Por ejemplo, si tomamos como referencia positiva el giro del rotor hacia la derecha, cuando la carga intenta desplazar el rotor hacia la misma derecha, el par estático aparece como negativo para compensar el sentido de la fuerza. Sin embargo, si la carga hace girar el rotor hacia la izquierda, el par estático aparece como positivo para tirar del rotor hacia la derecha.

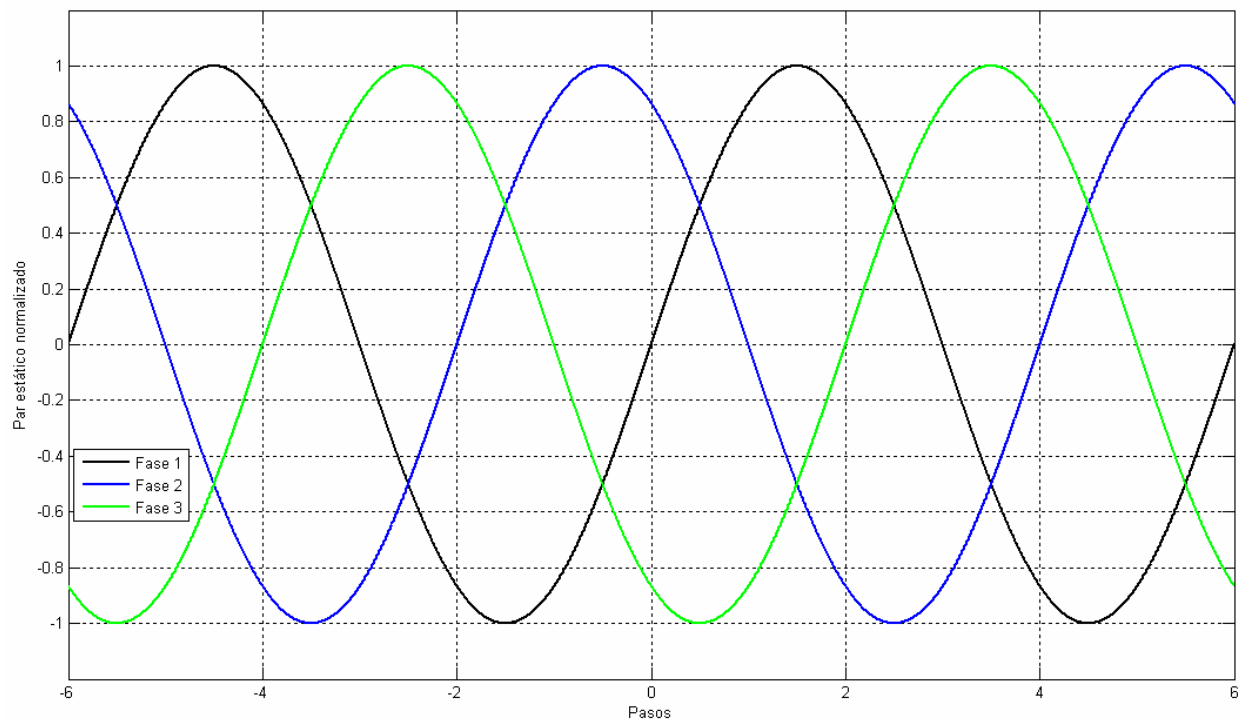


Fig. 11: Gráfico del par estático normalizado de un motor a pasos con 3 fases y la fase 1 como referencia o paso cero.

En la *Fig. 11* (véase también *Anexo A.*) se puede observar perfectamente los puntos de reposo (par igual a cero) que se corresponden con los pasos estables del motor. Si nos situamos en el paso cero vemos que está activa la fase uno ya que tiene par estático igual a cero y conforme vamos haciendo pasos a la derecha, se van activando las fases 2 y 3 respectivamente hasta volver a la uno.

Para determinar el error de paso que produce una cierta carga en el eje del rotor, podemos despejar el ángulo de la ecuación 2.3 y saber cuántos grados nos va a desplazar el rotor de la posición de paso la carga del motor.

$$\theta_{err} = \frac{2 \cdot n \cdot \theta_p}{2 \cdot \pi} \cdot \arcsen\left(\frac{T_L}{T_0}\right) \quad (2.7)$$

En donde:

T_L es el par estático provocado por la carga.

T_0 es el par estático máximo.

θ_p es el ángulo de paso del motor.

θ_{err} es el ángulo de error en la posición del rotor provocado por la carga.

n es el número de fases del motor.

De esta forma podemos determinar el error que nos proporcionará una cierta carga en nuestro sistema y rectificar, si es necesario, la posición del rotor.

- **Par dinámico (*dynamic torque* o *pull-out torque*)**

El par dinámico o *pull-out torque* es la característica no estática más importante de los motores a pasos. Representa el máximo par que el motor puede dar a una determinada frecuencia de excitación y valor de alimentación sin que el rotor pierda el sincronismo, es decir, sin perder pasos. Este parámetro se acostumbra a dar de forma gráfica con las curvas características y que dependerán del circuito de excitación y control del motor.

Generalmente un motor a pasos entrega el mayor par en condiciones estáticas. A medida que la velocidad va aumentando se reduce el par. De hecho, este fenómeno también ocurre en otro tipo de motores como por ejemplo los de continua, pero ahí el estudio de este fenómeno es diferente. El par capaz de entregar el motor es la fuerza aplicada sobre el rotor y es proporcional a la corriente de excitación de los bobinados de las fases.

Recordemos que en un circuito inductivo como los devanados de un motor a pasos, la inductancia se opone siempre a los cambios de corriente. Por eso, si aumentamos el valor de la frecuencia de conmutación, la corriente que circulará por cada fase del motor alcanzará un valor inferior debido a la oposición de la inductancia al cambio de corriente y al menor tiempo de inyección de la misma.

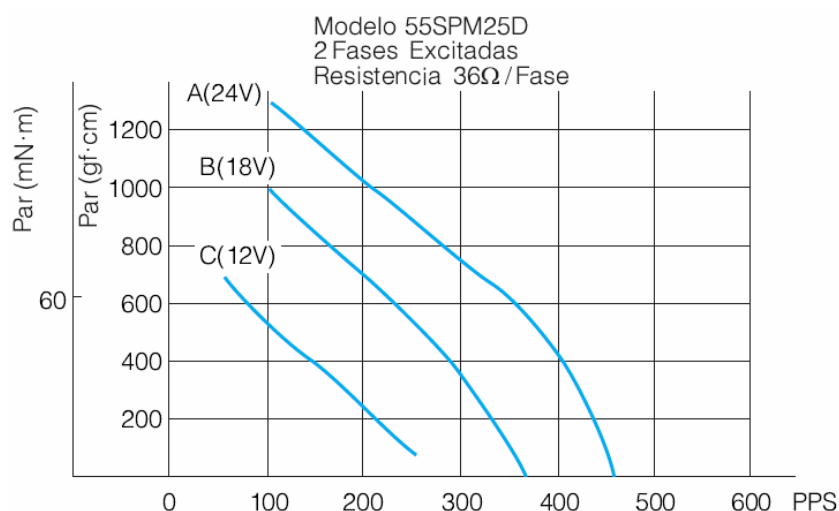


Fig. 12: Gráfico del par dinámico del motor a pasos 55SPM25D utilizado en este proyecto.

En la Fig. 12, se puede ver el gráfico del par dinámico del motor a pasos utilizado en este proyecto en función de la tensión de alimentación y la velocidad de conmutación entre pasos. Se observa que a bajas velocidades entre pasos el motor entrega su máximo par y, dependiendo del voltaje aplicado sobre sus fases, el par disminuye impidiendo la entrega del máximo valor. Si rebasamos la velocidad límite el motor no es capaz de entregar el suficiente par para mover el rotor y es posible que se pierdan pasos.

Una limitación de estas curvas es que se asume una velocidad constante. De hecho, la velocidad de giro del eje del motor de un motor a pasos nunca es constante ya que fluctúa entre pasos. De todas formas, la aproximación es válida en el caso de velocidades altas del orden de centenares de pasos por segundo.

En este proyecto, la velocidad de pasos no es un factor limitante ya que el tiempo de giro del motor lo marca el mayor retardo dentro de la cadena que, en este proyecto, se corresponde con la obtención de la matriz S por parte del analizador de redes.

Aproximadamente se tarda 1,5 seg. en obtener la matriz S en un margen frecuencial de 5GHz y con 200 puntos de medida. Por lo tanto, nunca se tarda menos de 1,5 seg. en dar un paso. Esta medida se corresponde con una velocidad de paso de 0,66 pasos por segundo. A esta velocidad de giro, el par dinámico entregado es máximo y disponemos de toda la fuerza del motor.

Puede darse el caso de tener mínimos relativos del par dinámico a bajas frecuencias de trabajo. Este fenómeno es debido a la resonancia mecánica del motor, y cuando se excita a una velocidad próxima a la velocidad de resonancia puede disminuir la entrega de par al eje del rotor, aunque no es el caso en este proyecto.

○ Par dinámico de arranque (*pull-in torque*)

El par dinámico de arranque o *pull-in torque*, determina el par máximo en que el motor es capaz de arrancar y parar instantáneamente sin perder pasos. Este parámetro es importante cuando tenemos una carga inercial que produce un elevado valor de fuerza al acelerarla y frenarla. Es decir, nos limita la velocidad de pasos por segundo para una determinada carga que aplica cierto par sobre el eje del motor.

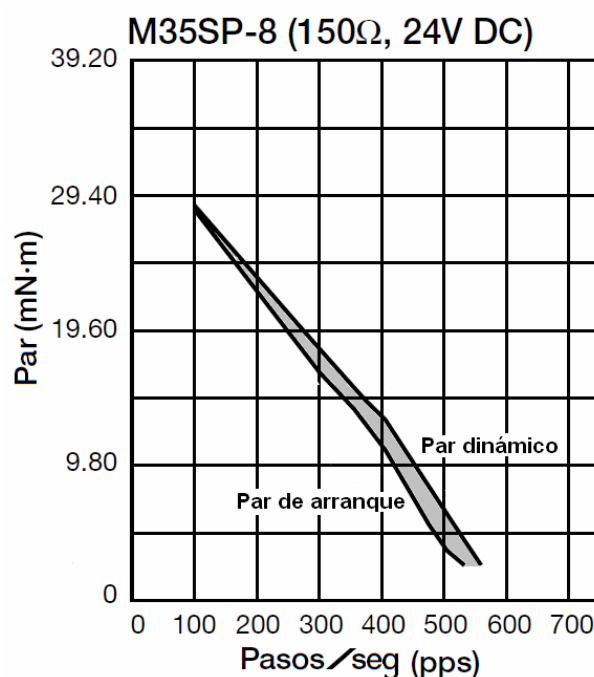


Fig. 13: Gráfico del par dinámico y par dinámico de arranque del motor a pasos M35SP-8.

Como puede verse en la Fig. 13, el gráfico del par dinámico es mayor que el par dinámico de arranque debido a que en el primero no se tiene en cuenta la posible inercia ejercida por la carga en el eje del rotor. Entonces, si sabemos que una cierta carga nos genera un cierto par al acelerarla o frenarla, este par es el que nos limita la velocidad de trabajo. A mayor par ejercido por la inercia de la carga, menor velocidad de trabajo.

La región comprendida entre el par dinámico y el de arranque se denomina “*slew region*” y el motor puede girar ya que tiene suficiente fuerza para hacerlo pero no puede acelerar o frenar la carga sin perder sincronismo, es decir, sin perder pasos. Siempre es recomendable trabajar por debajo del par dinámico de arranque para evitar problemas de sincronismo, pero si no es posible se puede trabajar en la zona de *slew* haciendo rampas de aceleración y desaceleración, normalmente desarrolladas en el software de control.

○ Par de parada

El par de parada es el par que ofrece el motor cuando no tiene ninguna fase alimentada y normalmente tiene valores más bajos que el par estático. Sólo existe par de parada en aquellos motores a pasos que tienen un imán permanente. En los motores de reluctancia variable, por ejemplo, el par de parada es cero y sólo es necesario que lo tengan en aquellas aplicaciones que sea imprescindible que el rotor esté quieto cuando no haya alimentación.

2.1.7. Clasificación de los motores a pasos según su excitación

El proceso de hacer girar un motor a pasos a derecha e izquierda, exige un cambio en la dirección del flujo magnético de forma independiente en cada una de las fases. La manera de cambiar este flujo es mediante un cambio en el sentido de la corriente inyectada en las fases, que principalmente se lleva a cabo utilizando dos esquemas: unipolar o bipolar. Cabe destacar que un motor a pasos puede tener devanados unipolares, bipolares o que puedan trabajar de las dos formas realizando unas pequeñas conexiones externas en el cableado del motor. Aún así, los circuitos de excitación son muy diferentes según se utilice un motor unipolar o bipolar.

○ Motor unipolar

En un motor unipolar cada fase está formada por un devanado independiente o también se pueden crear dos fases de un mismo devanado con una conexión en su punto medio. El cambio en la dirección del flujo se realiza mediante la conducción de la otra mitad del devanado, en el caso de estar dividido por la mitad, o por la conducción del otro devanado, en caso de ser independientes.

Conexiones motor Unipolar

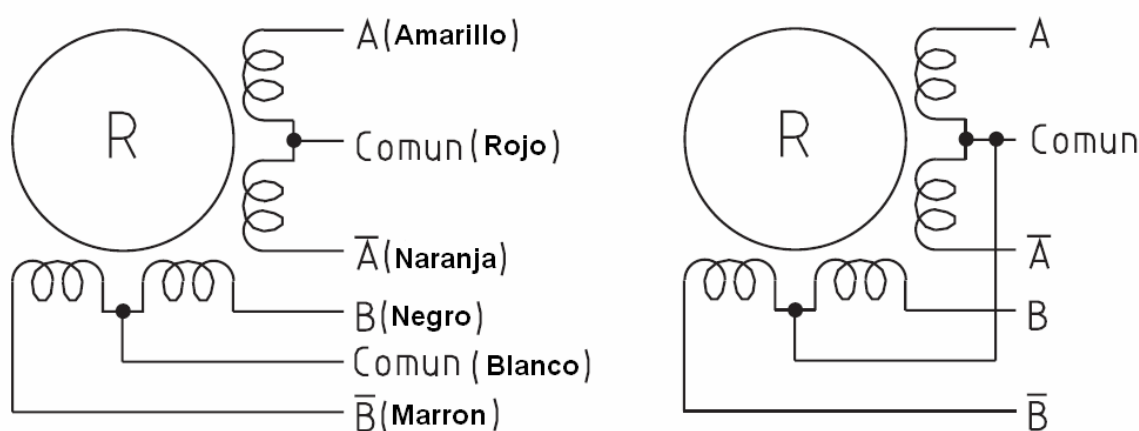


Fig. 14: Motor unipolar con dos devanados divididos por la mitad. A la izq. 6 conexiones con su color habitual de cableado y a la dcha. se han unido los puntos medios de los dos devanados para obtener 5 conexiones.

En la Fig. 14, se puede observar el esquema de un motor unipolar de dos devanados divididos por la mitad para realizar el cambio de sentido del flujo magnético obteniendo 4 fases. A la izquierda se observa que no se han conectado los puntos medios de los devanados mientras que a la derecha se han conectado para disminuir el cableado externo. Normalmente en el común se conecta la referencia de voltaje o la alimentación positiva dependiendo del *driver* utilizado.

El principal inconveniente de este tipo de motores es que sólo se puede utilizar la mitad de los devanados, razón por la cual el motor unipolar siempre ofrece menos par dinámico que un motor bipolar de la misma medida.

La ventaja del motor unipolar radica en que el circuito eléctrico de excitación es más sencillo ya que sólo necesita un interruptor por fase (dos por devanado) frente a los cuatro por devanado de los bipolares.

Del esquema anterior se puede llegar a la conclusión que si no tenemos conectados los puntos medios de los devanados, se puede convertir un motor unipolar en bipolar si dejamos libre este mismo punto medio de cada devanado. Es decir, que un motor unipolar es un motor bipolar al cual se le ha insertado una conexión en el punto medio de cada devanado.

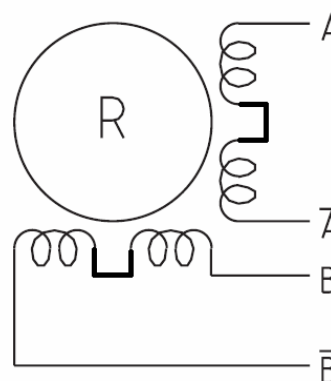


Fig. 15: Motor unipolar trabajando como bipolar.

Los motores unipolares han sido los más utilizados a lo largo de la historia y el motivo principal ha sido que el par no es un factor decisivo en el diseño y se prefiere una circuitería más sencilla. Además, el coste de un circuito bipolar podía llegar a ser muy elevado respecto al de un unipolar. A pesar de todo, hoy en día y gracias a la facilidad de integración, la diferencia de costes entre las dos circuiterías no es mucha y se tiende a la elección del motor en función del trabajo a desarrollar y no por los costes derivados de la circuitería.

○ Motor Bipolar

En un motor bipolar, el cambio de dirección del flujo magnético para hacer rotar el motor a izquierda o derecha, se hace mediante el cambio de polaridad de la corriente aplicada en las fases o devanados. Normalmente, para cambiar esta polaridad de la corriente inyectada en los devanados, hacen falta cuatro interruptores por fase situados en puente en H.

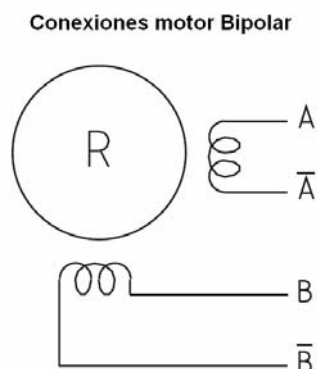


Fig. 16: Motor bipolar de dos devanados.

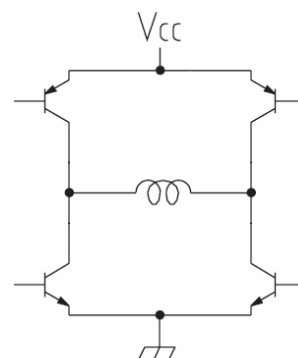


Fig. 17: Interruptores necesarios para activar un devanado en los dos sentidos de giro.

En la Fig. 16 podemos ver el esquema de un motor bipolar con dos devanados y por lo tanto dos fases. Dependiendo de los niveles de excitación de A y B, podemos hacer circular la corriente en cierto sentido o en sentido contrario logrando un cambio en el sentido del flujo magnético y por lo tanto un giro en sentido positivo o negativo del rotor.

El inconveniente, tal y como se muestra en la Fig. 17, es que para hacer circular la corriente en ambos sentidos, tenemos que disponer de cuatro interruptores o transistores por devanado. Si no se coordinan correctamente estos cuatro interruptores, podemos tener problemas de sobreintensidad o cortocircuitos. Para evitar todos estos problemas, existen integrados como el L293 que partiendo de señales TTL activan las distintas fases del motor bipolar.

2.1.8. Clasificación de los motores a pasos según su construcción

Dependiendo de la construcción del motor a pasos, podemos clasificarlos según tres grandes grupos: motores de reluctancia variable, motores permanentemente magnetizados e híbridos. A continuación se describen sus principales características y su modo de funcionamiento.

○ Motores de reluctancia variable

Los motores de reluctancia variable no tienen ninguna parte permanentemente magnetizada; ni el rotor ni el estator. Simplemente el rotor, que suele estar construido de hierro dulce, se mueve bajo el principio de mínima reluctancia, es decir, se orienta según ofrece menor resistencia al paso del flujo magnético creado por los devanados del estator al ser excitados por una corriente. Típicamente hay un estator simple o múltiple con diversos devanados y con un determinado número de polos. El rotor suele ser un cilindro de hierro dulce no magnetizado formado por uno o varios discos adyacentes,

dependiendo de si el estator también es simple o múltiple. Suelen tener pequeñas cámaras de aire situadas de forma axial o radial para facilitar la refrigeración.

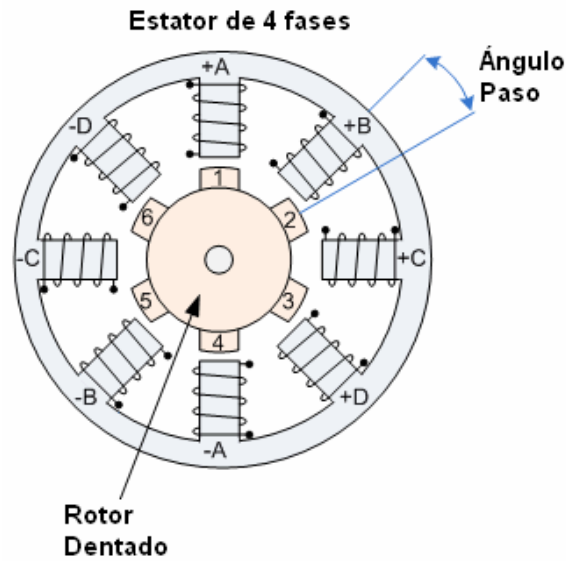


Fig. 18: Sección de un motor a pasos de reluctancia variable.

La Fig. 18 se trata de un motor de reluctancia variable con estator y rotor simple. El estator tiene cuatro fases (A, B, C y D) con sus respectivos polos positivos y negativos. El rotor es de 6 polos no magnetizados y se orientan según el principio de mínima reluctancia. En la figura el rotor está alineado con la fase A, y al dar un paso y activarse la fase B, el rotor orientará el polo número 2 con la fase B haciendo girar el rotor en sentido antihorario un ángulo de paso.

Los motores de reluctancia variable y estator simple son los más compactos y sencillos de construir aunque sólo pueden conseguir ciertos valores de ángulo de paso comprendidos normalmente entre 30° y 1.8° . Suelen tener un cierto número de dientes en cada fase del estator y otro en cada polo del rotor y es por este motivo que se pueden conseguir diferentes ángulos de paso con el mismo número de fases y polos.

Como se muestra en la siguiente ecuación, sabiendo el número de dientes en el estator y en el rotor, podemos encontrar el número de pasos por vuelta:

$$N = \left| \frac{N_r \cdot N_s}{N_r - N_s} \right| \quad (2.8)$$

En donde:

N_r es el número de dientes en el rotor.

N_s es el número de dientes en el estator.

En la *Fig. 18* tenemos un diente por polo y un diente por fase. Así, aplicando la ecuación anterior obtendríamos 24 pasos por vuelta o equivalentemente 15° por paso.

Existen también los motores de reluctancia variable y estator múltiple que no son otra cosa que encadenar múltiples secciones independientes de estatores y rotores simples desplazados una cierta porción de la anchura de los dientes. Es decir, si tenemos 3 estatores, la distancia desplazada entre cada estator y rotor es un tercio de la distancia entre dientes adyacentes.

Los motores de estator múltiple han sido concebidos para ofrecer mejores prestaciones y un mayor par de salida. Aunque son más caros de fabricar, los motores de estator múltiple ofrecen mejor rendimiento y consiguen velocidades muy altas del orden de 30.000 rev/min.

○ Motores de imán permanente

De motores paso a paso de imán permanente existen principalmente dos tipos: los de rotor magnetizado y los de estator magnetizado.

En los motores con estator permanentemente magnetizado, el estator tiene dos partes diferenciadas: los imanes permanentes y las fases. A pesar de su gran par y su fácil fabricación, estos motores son poco usados ya que sólo permiten girar en un único sentido el rotor y las velocidades son bastante bajas.

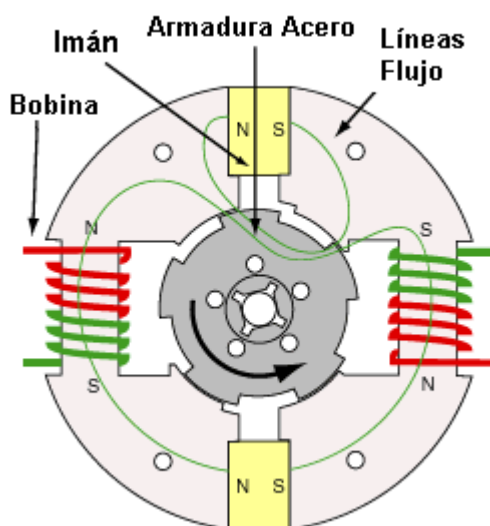


Fig. 19: Motor con estator magnetizado permanentemente.

Como se puede observar en la *Fig. 19*, cuando se aplica una corriente en las fases del estator, se modifica el flujo magnético que circula tanto por el estator como por el rotor creándose una alteración de las líneas de flujo en el imán permanente de arriba. Esto hace que el rotor gire para buscar la mínima oposición a las líneas de flujo magnético y orienta sus polos. Cuando se invierte el sentido de la corriente por las fases, cambia el flujo magnético provocando la alteración en el imán de abajo. Entonces el rotor vuelve a girar para situar sus polos en la posición de menor resistencia al flujo.

El otro tipo de motores, los motores con rotor permanentemente magnetizado, o también llamados “*tin can*” o “*canstack*”, son motores de bajo coste que ofrecen entre 24 y 48 pasos por vuelta. Su principal característica es que el rotor se encuentra permanentemente magnetizado con polos norte y sur alternativamente situados en líneas paralelas al eje de rotación y pueden encontrarse en el mismo cilindro del rotor o en dos cilindros separados.

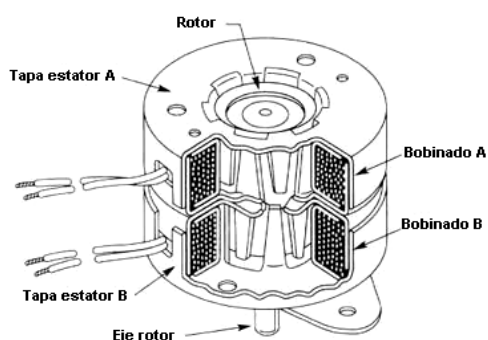


Fig. 20: Motor con rotor permanentemente magnetizado.

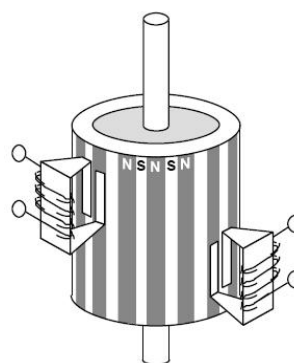


Fig. 21: Representación simplificada del modo de trabajo de un motor con rotor permanentemente magnetizado de 7.5° por paso.

En la *Fig. 20* se puede ver un motor con rotor permanentemente magnetizado con dos devanados. Al circular la corriente por estas bobinas, se crean polos magnéticos según la disposición de las mismas, logrando una alternación de polos norte y sur. Entonces, el rotor, que se encuentra permanentemente magnetizado, se orienta según el flujo magnético.

En la *Fig. 21* se observa una representación simplificada del modo de trabajo de los motores con el rotor magnetizado. En el rotor se observan los pares de polos norte y sur alternativamente que se orientan hacia los dientes de las fases del estator según circule su corriente.

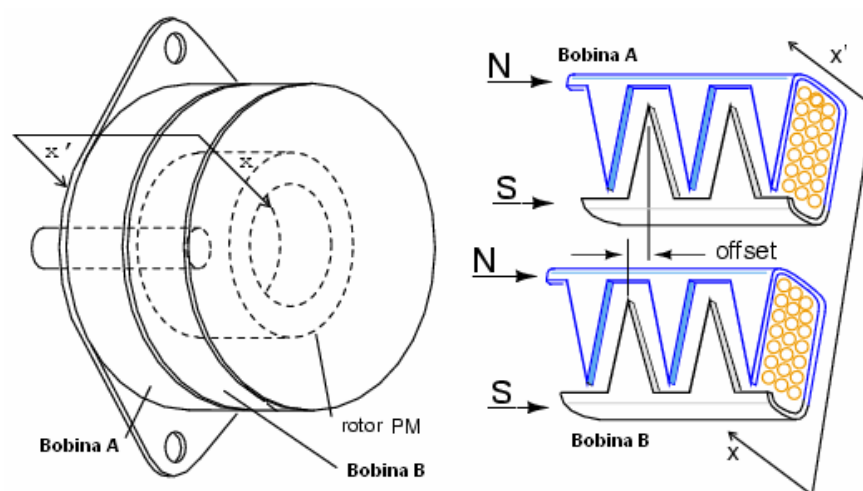


Fig. 22: Detalle del estator de un motor con el rotor permanentemente magnetizado.

Como podemos ver en la Fig. 22, los dos devanados tienen parejas de polos separados entre ellos la mitad de la distancia de dos dientes consecutivos. Y los dos devanados 1 y 2 se encuentran separados un *offset* que se corresponde con un cuarto de la distancia entre dientes. El rotor no suele tener dientes como sucede con los motores de reluctancia variable. Cuando aplicamos tensión en los devanados de forma alternativa, el rotor gira $\frac{1}{4}$ de la distancia entre dientes logrando avanzar un paso.

Este tipo de motores con un devanado por bobina y una conexión en medio de cada devanado logrando dos fases (conexión unipolar) es el más utilizado hoy en día en la industria, ya que ofrece un bajo coste de fabricación tanto del motor como del circuito electrónico de control externo.

Para terminar con este tipo de motores, decir que el motor de este proyecto, es un motor de este tipo, con rotor permanentemente magnetizado con estator de 4 fases unipolar y de 7.5° de paso.

○ Motores híbridos

Este tipo de motores son una mezcla entre los de reluctancia variable y los permanentemente magnetizados ya que tiene el rotor parcialmente magnetizado. Fueron concebidos para funcionar como motores síncronos de baja velocidad, y de hecho, pueden ser activados con una fuente de dos fases de corriente alterna aunque su principal uso es con fuentes digitales.

El estator está compuesto por varias fases magnetizadas eléctricamente según la secuencia exterior de excitación. El rotor está formado por tres partes: un cilindro con dientes polarizadas como norte, otro igual pero polarizado como sur desplazado medio paso de diente respecto al cilindro norte, y el eje que los une y a la vez los polariza.

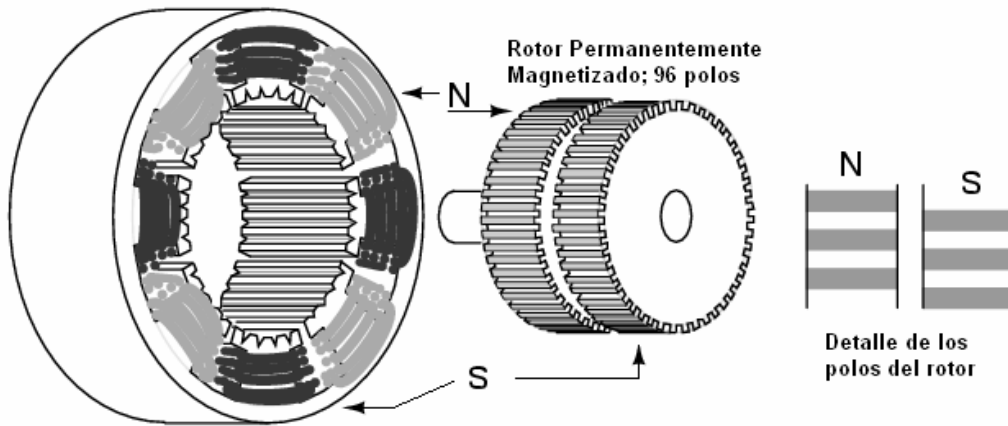


Fig. 23: Esquema representativo de un motor a pasos híbrido.

Como se puede ver en la Fig. 23, el estator tiene 8 polos y 5 dientes por polo. El rotor dispone de dos cilindros, uno con 96 polos norte y otro con 96 polos sur con un desfase entre ambos de medio paso de diente.

Cuando tenemos una fase activa, ésta atrae hacia sus dientes los polos del rotor correspondientes a la polarización inversa creada por la corriente que la atraviesa. Es decir, si se crea un polo norte en la fase del estator, atrae los dientes con polo sur del rotor. Si posteriormente activamos la fase adyacente con polarización inversa, atraerá a los polos norte del rotor, avanzando medio paso de diente.

Para calcular el ángulo de paso, utilizamos la siguiente ecuación:

$$\theta_p = \frac{360^\circ}{N_s} - \frac{360^\circ}{N_r} \quad (2.9)$$

En donde:

N_r es el número de dientes en el rotor.

N_s es el número de dientes en el estator.

Así, para el ejemplo de la Fig. 23, tenemos un rotor con 96 dientes o polos y un estator con 40 dientes. Aplicando la anterior fórmula, cada paso equivale a 5.25° de rotación del rotor o equivalentemente 68.5 pasos por vuelta.

Para terminar con el capítulo de la clasificación de motores, destacar que existen muchos otros motores considerados a pasos pero que no son de importancia para este proyecto.

Existen los motores llamados “*solenoid-ratched*” que podríamos traducirlo como rueda dentada con solenoide. Se trata del motor a pasos más simple pero que solamente puede girar en una dirección y tiene que ser activado por una única fase que actúa sobre una bobina que en su interior tiene un núcleo imantado que puede desplazarse libremente. Este imán está conectado a una varilla que actúa sobre los dientes de la rueda dentada. Cuando se alimenta la fase, la bobina crea un campo magnético que hace desplazar el imán provocando el avance de la rueda dentada mediante la varilla.

También encontramos motores lineales de reluctancia variable o de imán permanente. Provocan un desplazamiento lineal y dependiendo del número de fases del estator, pueden dar más o menos pasos a derecha e izquierda.

Para terminar, comentar que existen los motores paso a paso electrohidráulicos, que no dejan de ser un motor a pasos eléctrico convencional pero con un amplificador de par hidráulico. De hecho, este tipo de motores se usan en aplicaciones con gran demanda de par y así compensar la ineficiencia de los motores paso a paso solamente eléctricos.

2.2. El filtro de microondas

La finalidad de este proyecto es la sintonización automática de la frecuencia de resonancia de un resonador de microondas que disponga de un tornillo de sintonización. Pero antes de empezar el análisis del proyecto en sí, se va a hacer un resumen sucinto sobre las principales características de los filtros de microondas para entender mejor su funcionamiento.

2.2.1. Definición del filtro de microondas

Un filtro de microondas es un bipuerto pasivo que se encarga de atenuar ciertas frecuencias y dejar pasar otras en función de la respuesta frecuencial que posea el filtro. Es bipuerto porque tiene un puerto de entrada y uno de salida con la señal filtrada, y es pasivo porque no tiene elementos que añadan potencia. Por tanto, la potencia de entrada puede ser transmitida a la salida o disipada.

Existen multitud de filtros de microondas y dependiendo de la frecuencia de trabajo hay diferentes formas de construirlos. Los sistemas de construcción más comunes son: mediante líneas de transmisión planares (*microstrip*), líneas acopladas coplanares, guías de ondas rectangulares, guías de ondas rectangulares con diafragmas, estructuras resonantes encadenadas, etc. También es posible clasificarlos teniendo en cuenta su respuesta frecuencial, entre los cuales destacan: resonador simple, filtro paso bajo, filtro paso alto, filtro paso banda, filtro de banda eliminada, etc. Y en función de esta respuesta existen diferentes formas de implementarlos: Butterworth, Chebyshev, elíptico, etc.

2.2.2. Matriz S

La matriz de *Scattering* (dispersión) o también llamada matriz S , define la relación entre las amplitudes complejas de las ondas de tensión incidentes y reflejadas en los planos de referencia de los puertos. En una red de p puertos, dichos parámetros se agrupan en una matriz de *scattering* $S \in \mathbb{C}^{p \times p}$,

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1p} \\ S_{21} & S_{22} & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \cdots & S_{pp} \end{bmatrix} \quad (2.10)$$

De modo que si las amplitudes de las ondas incidentes y reflejadas se representan respectivamente como componentes de los vectores \bar{a} y $\bar{b} \in C^p$, dicha relación viene dada por la ecuación siguiente:

$$\bar{b} = S \cdot \bar{a} \quad (2.11)$$

Con:

$$\bar{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} \quad \bar{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \quad (2.12)$$

Así, se puede expresar cada parámetro de la matriz S como la relación de energía que entra por la puerta i y sale por la puerta j de la siguiente forma:

$$S_{ji} = \left. \frac{b_j}{a_i} \right|_{a_k=0, \forall k \neq i} \quad (2.13)$$

En donde $a_k=0$ significa que la puerta k está terminada con una impedancia igual a la de referencia de la puerta y no se refleja energía en la carga de vuelta hacia el dispositivo. Es decir, que toda la energía saliente por la puerta k es absorbida por la carga adaptada.

Así, se puede decir que los coeficientes de la matriz S son coeficientes de transmisión entre puertas y dan una idea de la energía que sale por una puerta si alimentamos otra cuando los demás accesos están adaptados. Una vez todos los parámetros S están calculados, cualquier salida se encuentra como combinación lineal de todas las entradas y se puede ver el circuito como una caja negra y calcular las salidas b en función de las entradas a y los parámetros S .

Pero, para relacionar las entradas y salidas a y b con parámetros físicos característicos de los circuitos (tensión, corriente e impedancia), deben de aplicarse las siguientes igualdades:

$$a_i = \frac{1}{2} \left(\frac{V_i}{\sqrt{Z_{0i}}} + \sqrt{Z_{0i}} \cdot I_i \right) \quad (2.14)$$

$$b_i = \frac{1}{2} \left(\frac{V_i}{\sqrt{Z_{0i}}} - \sqrt{Z_{0i}} \cdot I_i \right) \quad (2.15)$$

En donde:

V_i es la tensión en el acceso i .

I_i es la corriente en el acceso i .

Z_{0i} es la impedancia característica del acceso i .

Así y sin entrar en más particularidades, se puede trabajar con un circuito de microondas sin saber su composición interna. Solamente disponiendo de la matriz S ya se puede calcular la cantidad de energía que saldrá por una puerta al alimentar las otras.

2.2.3. Matriz S del filtro de microondas

Como se ha mencionado anteriormente, el filtro de microondas es un bipuerto pasivo. Como solamente tiene dos accesos, su matriz S se compone de cuatro parámetros:

$$S = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \quad (2.16)$$

Y como es pasivo, se cumple que:

$$|S_{11}|, |S_{12}|, |S_{21}|, |S_{22}| \leq 1 \quad (2.17)$$

Si tomamos el puerto 1 como entrada del filtro y el puerto 2 como salida, entonces el parámetro más importante es el S_{21} ya que nos da la respuesta frecuencial del filtro. Este parámetro es llamado comúnmente pérdidas de inserción, haciendo referencia a las pérdidas de energía que se producen a la salida al inyectar señal por la entrada.

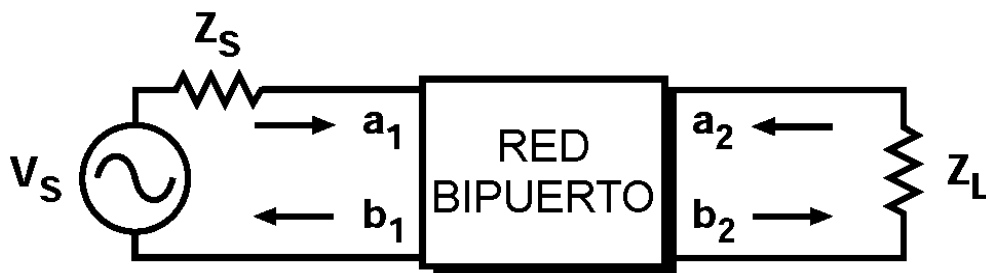


Fig. 24: Red bipuerto.

Como se puede ver en la Fig. 24, se trata de un bipuerto con una fuente de tensión V_S de impedancia Z_S y una carga en el puerto de salida Z_L . También se muestran las ondas progresivas a y las regresivas b para cada acceso.

Si se inyecta un cierto nivel de energía en la entrada a una cierta frecuencia, obtendremos el nivel de energía saliente a esa misma frecuencia. Ahora bien, si se hace un barrido con múltiples frecuencias, obtendremos la respuesta frecuencial del filtro en el rango de frecuencias barrido.

Esto es justamente lo que hace el analizador de redes. Sabiendo el nivel de energía inyectado en la entrada del filtro, mide el nivel de energía saliente y nos grafica la porción de energía que sale en función de la que entra expresada en dB y para el rango de frecuencias dado. Más adelante, en el apartado 3.1 *El analizador de redes*, se explica con más detalle el funcionamiento del mismo.

El parámetro S_{11} o también llamado pérdidas de retorno, no es menos importante ya que nos da una idea de la cantidad de energía que se refleja en el puerto de entrada y no se inyecta en el filtro. Para un correcto funcionamiento del filtro, es importante que este valor sea lo más pequeño posible en la banda de paso, ya que indicaría que casi toda la potencia inyectada a la entrada, entra realmente en el filtro y se transmite, o no, a la salida.

El parámetro S_{22} nos da una idea de la cantidad de energía incidente en el puerto de salida y que “rebota” en este mismo puerto sin entrar en el dispositivo. Este parámetro interesa que sea muy bajo ya que así no se provoca inestabilidad en la carga ni transitorios demasiado grandes o casi “permanentes” si el circuito empezara a oscilar.

2.2.4. Ancho de banda

Para señales analógicas, el ancho de banda es la longitud, medida en Hz, del rango de frecuencias en el que se concentra la mayor parte de la potencia de la señal. También son llamadas frecuencias efectivas las pertenecientes a este rango. Así, el ancho de banda de un filtro son las frecuencias en las que su atenuación al pasar a través del filtro se mantiene igual o inferior a 3 dB comparada con la frecuencia de pico máximo (f_0).

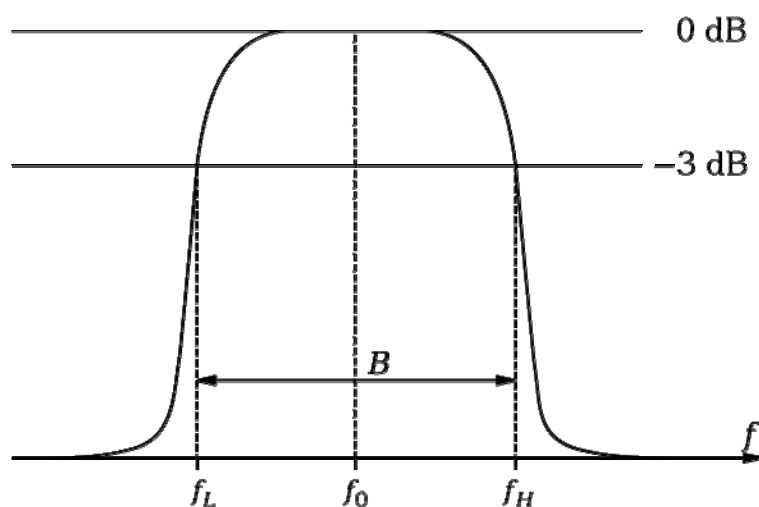


Fig. 25: Ancho de banda de un filtro paso banda.

Como se puede ver en la *Fig. 25*, el ancho de banda son todas las frecuencias comprendidas entre las frecuencias f_L (frecuencia de corte inferior) y f_H (frecuencia de corte superior), que son las frecuencias en las cuales la amplitud de la señal saliente del filtro cae a la mitad respecto al máximo (3dB). Así, podemos calcular el ancho de banda como:

$$BW = f_H - f_L \quad (2.18)$$

2.2.5. Frecuencia central

La frecuencia central de un filtro de microondas es la frecuencia central de la banda de paso y es la frecuencia a la que se dice que está centrado el filtro. Para calcular la frecuencia central se procede de la siguiente forma:

- Primero se busca la frecuencia con la mínima atenuación (f_0) y el valor de su atenuación.
- Luego, al valor de la atenuación de la frecuencia de mínima atenuación (f_0) se le resta 3dB y se encuentra la frecuencia de corte inferior (f_L) y superior (f_H). En este punto tenemos el ancho de banda del filtro.
- Para encontrar la frecuencia central, buscamos el punto medio del ancho de banda :

$$f_c = \frac{f_H + f_L}{2} \quad (2.19)$$

2.2.6. Resonador utilizado en el proyecto

Para comprobar el correcto funcionamiento del software de control diseñado, así como del hardware, firmware y la estructura de soporte, se precisa un resonador de microondas sintonizable en frecuencia. Como solamente se pretende sintonizar un parámetro, la frecuencia de resonancia, se requiere de un resonador que en su respuesta frecuencial pueda observarse de forma clara y precisa el cambio de la frecuencia de resonancia.

Así que siguiendo las premisas anteriores, se ha tomado como punto de partida el resonador débilmente acoplado mostrado en la *Fig. 26*. El resonador en cuestión es una cavidad hecha en cobre que en su interior alberga un resonador de cuarto de onda realizado también en cobre y sus respectivas líneas de alimentación. La frecuencia inicial de resonancia es de 9,42GHz tal y como puede verse en la *Fig. 27*.

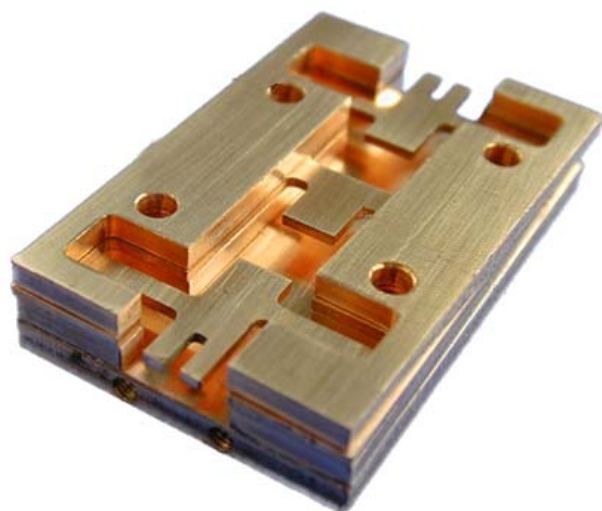


Fig. 26: Vista interior del resonador débilmente acoplado utilizado.

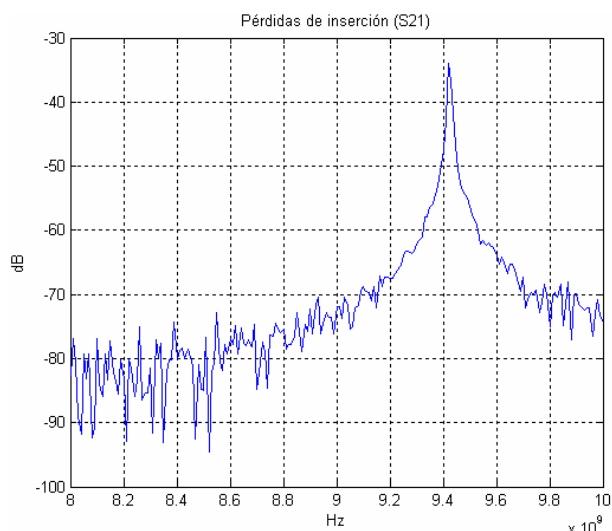


Fig. 27: Pérdidas de inserción del resonador débilmente acoplado.

Como puede verse en la Fig. 28, el resonador tiene la máxima densidad de campo magnético cerca de los cortocircuitos y el campo eléctrico máximo en el lado opuesto. Por lo tanto, configurando adecuadamente este resonador se pueden conseguir acoples eléctricos, magnéticos o mixtos. Si se colocan los cortocircuitos uno enfrente del otro se consigue un acople magnético pero si hacemos lo mismo con los extremos abiertos, se consigue un acople eléctrico.

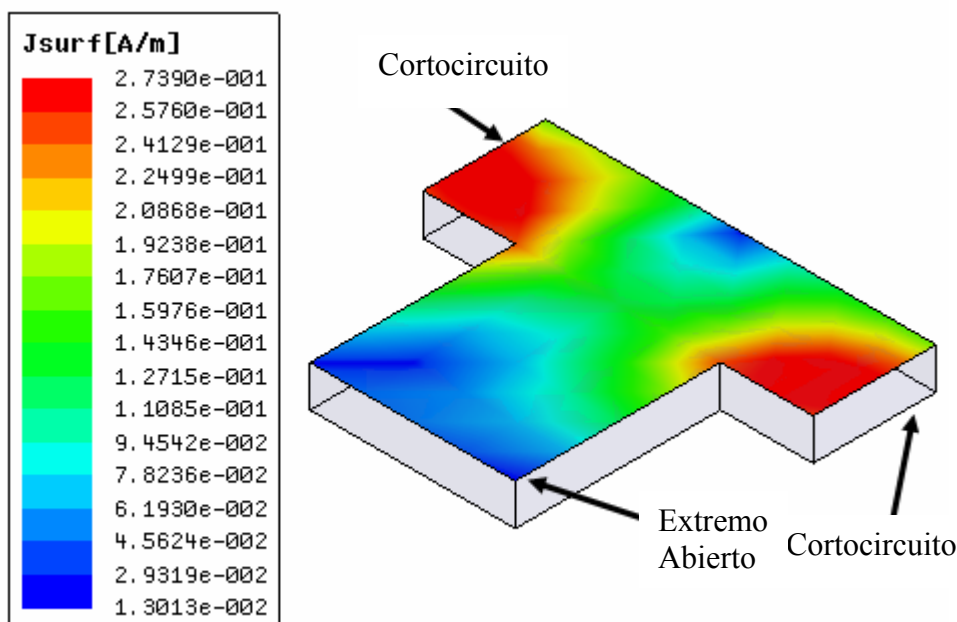


Fig. 28: Distribución superficial de corriente en el resonador de cuarto de onda.

Pero para lograr un cambio en la frecuencia central del resonador, se ha perforado un orificio en la tapa superior y se ha insertado un tornillo de sintonización. Cuando este tornillo penetra dentro de la cavidad ocasiona perturbaciones en el campo electro magnético provocando el cambio de la frecuencia de resonancia. Cuanto mayor es la perturbación ocasionada por el tornillo, mayor rango de sintonía. Así que tras un análisis del resonador, se ha insertado un tornillo de 5mm de diámetro métrico estándar.

Al ser débilmente acoplado, se tenían unas pérdidas de inserción de unos -30dB que a frecuencias inferiores bajaban hasta los -50dB y se confundían con el ruido, de unos -60dB.

Así que para disminuir estas pérdidas, se alargaron las líneas de alimentación hacia el resonador central unos 5,5mm. De esta forma se consiguió aumentar hasta los -8dB las pérdidas de inserción y lograr un rango de sintonía de 3GHz.

En la Fig. 29 se muestran las modificaciones realizadas en el resonador débilmente acoplado. El círculo azul muestra la zona de perturbación del tornillo de sintonización, y las superficies azules se corresponden con las extensiones realizadas a las líneas de alimentación para disminuir las pérdidas de inserción.

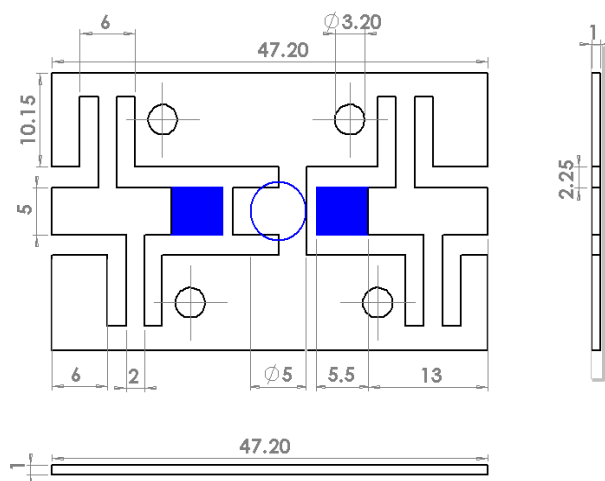


Fig. 29: Modificaciones realizadas sobre el resonador original.

Tras las modificaciones, el resonador consigue sintonizar frecuencias desde los 5,9GHz hasta los 8,9GHz con unas pérdidas de inserción de -8dB aproximadamente en todo el rango de frecuencias.

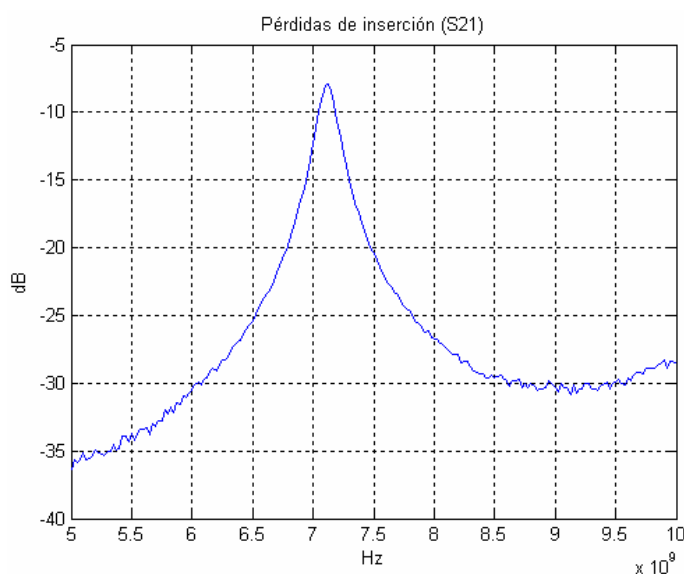


Fig. 30: Pérdidas de inserción del resonador modificado a 7,125GHz.

Como puede verse en la *Fig.* , la respuesta obtenida después de las modificaciones es idónea para comprobar el correcto funcionamiento del proyecto ya que al desplazarse la frecuencia central del resonador, se desplaza el pico de las pérdidas de inserción. Así no hay otros factores que puedan oscurecer los resultados ni interferir con el objetivo del proyecto.

Gracias a las modificaciones se ha conseguido un rango de frecuencias sintonizables excelente, desde los 5,9GHz hasta los 8,9GHz, y unas pérdidas de inserción idóneas para ser medidas por el analizador de redes y poder corroborar el correcto funcionamiento de todo el proyecto.

Aunque no sea de gran importancia para el proyecto, cabe mencionar que el ancho de banda a -3dB va desde los 200MHz a frecuencias bajas (5,9GHz) hasta los 400MHz a frecuencias más altas (8,9GHz).

3. Instrumentación

En este proyecto se ha utilizado como instrumentación esencial el analizador de redes. Bien es cierto que también se ha utilizado un multímetro digital para medir niveles de tensión, corriente e impedancias, generadores de señal para alimentar el hardware, y el osciloscopio para capturar las señales del puerto serie. Pero su utilidad ha sido tan fugaz que no se consideran lo suficientemente importantes para dedicarles un apartado específico dentro de este informe y, al tratarse de instrumentos muy comunes, seguramente los lectores de este proyecto ya están familiarizados con ellos.

3.1. El analizador de redes

El analizador de redes se ha convertido en una de las herramientas más importantes a la hora de caracterizar componentes y dispositivos de alta frecuencia. Los nuevos analizadores vectoriales de red, pueden medir las componentes de una magnitud, fases y retardos de grupo, la impedancia de un puerto en una carta de Smith e incluso mostrar la distancia en la carta de Smith para una adaptación de impedancias de un puerto con impedancia característica diferente o un fallo en la línea de transmisión.

En este proyecto se ha trabajado con el analizador de redes HP8510. Se trata de un analizador de redes vectorial capaz de medir frecuencias comprendidas entre 45MHz y 50GHz, pudiendo alcanzar los 110GHz con los accesorios y cableados apropiados. Para más información acerca de este analizador de redes, se redirige al lector al *Anexo F* de este mismo documento.

3.1.1. Diferencias entre un analizador vectorial de redes y un analizador de espectros

Los analizadores de redes difieren en forma y funcionalidad de otras herramientas utilizadas comúnmente para el análisis de sistemas de comunicación y componentes de alta frecuencia como el analizador de espectros.

Con el analizador de espectros se pueden medir señales externas desconocidas. Por contrapartida, el analizador de redes se sirve de fuente de frecuencia sintetizada para proveer un estímulo de test conocido que permite barrer un rango de frecuencias o de potencias específico.

El analizador de espectros se utiliza generalmente para medir señales tales como el nivel de una portadora, bandas laterales, armónicos y ruido de fase. Normalmente se configuran en un solo canal y sin fuente. Este tipo de instrumento tiene un gran rango de anchos de banda de IF

disponibles que le permite analizar diferentes tipos de señales y usualmente son utilizados como fuentes externas para tests de estímulo/respuesta no lineal. Si se combinan con un generador de señales, los analizadores de espectros pueden usarse para muestrear componentes escalares y mostrarlas en frecuencia, pero nunca consiguen información sobre la fase.

Sin embargo, los analizadores de redes son capaces de medir una gran variedad de parámetros del dispositivo bajo medida (*DUT: Device Under Test*) incluyendo su magnitud, su fase o su retardo de grupo. Además, los analizadores de redes pueden servir como señales externas de estímulo, separan señales incidentes o reflejadas de un dispositivo, pueden utilizarse como receptores para la detección de señales y gracias a su procesador de circuitos muestran los resultados por pantalla con muchas posibilidades de visualización.

La separación de las señales permite medir una parte de la señal incidente y generar una referencia en el cálculo de los parámetros. Ello permite separar las señales incidentes (*forward*) y reflejadas (*reverse*) presentes a la entrada del DUT. El hardware diseñado para este propósito incluye divisores de potencia (resistivos y de espectro ancho pero con altas pérdidas de inserción), acopladores direccionales (que tienen pocas pérdidas pero usualmente limitados en banda), y puentes direccionales (que se utilizan comúnmente para medir señales reflejadas sobre un extenso ancho de banda, pero que pueden tener pérdidas significativas).

3.1.2. Estructura básica de un analizador de redes

El analizador de redes se utiliza comúnmente para medir los parámetros S de un circuito aunque también puede medir la variación de una magnitud y su fase en frecuencia, la desviación respecto una fase lineal, el retardo de grupo, etc.

El esquema interno de un analizador de redes se puede dividir en cuatro bloques:

- Procesador y pantalla para la revisión de resultados.
- Una fuente de estímulos.
- Dispositivos para separar las señales.
- Receptores que conviertan y detecten las señales.

A continuación se explica brevemente cada uno de los bloques que componen un analizador de redes.

○ Procesador y pantalla de visualización

El último bloque principal del analizador de redes es el procesador y la pantalla de visualización de resultados. Aquí es donde los parámetros de transmisión y reflexión toman forma para ser fácilmente interpretados. La mayoría de analizadores de redes tienen características similares como por ejemplo la escala lineal o logarítmica, gráficas

en coordenadas polares, representación en la carta de Smith, marcadores en la pantalla para posicionar el cursor, ajuste del rango de los ejes en la pantalla, etc.

○ Fuente

La fuente de señal suministra el estímulo necesario para medir la respuesta al estímulo de nuestro dispositivo, ya sea haciendo un barrido en frecuencia o en nivel de potencia inyectada. Normalmente estas fuentes se basan en un oscilador controlado por voltaje en lazo abierto (VCO) ya que es una solución barata. Si se desea un mayor rendimiento se pueden utilizar sintetizadores para realizar el barrido de frecuencia, especialmente para medir dispositivos de banda estrecha.

El exceso de ruido de fase de los VCO en lazo abierto, degrada considerablemente la precisión de la medida cuando se miden componentes de banda estrecha con valores de “*span*” pequeños.

○ Separación de señales

El hardware utilizado para separar las señales también es llamado “*test set*” y normalmente está integrado dentro del mismo analizador de redes. Hay dos funciones que el hardware de separación de señales debe de hacer. La primera es medir una parte de la señal incidente para proporcionar una referencia y posteriormente calcular las relaciones entre diferentes señales. Esta función puede realizarse mediante divisores normalmente resistivos (*splitter*), o acopladores direccionales.

Los divisores resistivos son dispositivos de banda ancha pero por contrapartida tienen unas pérdidas de 6dB o más en cada brazo.

Los acopladores direccionales tienen una baja pérdida de inserción (hacia el brazo principal) y muy buen aislamiento y directividad. Normalmente se usan en analizadores de redes de microondas pero su respuesta paso alto no los hace aptos para trabajar en frecuencias inferiores a los 40MHz.

La segunda función que tienen que desempeñar los separadores de señal es separar las ondas incidente y reflejada a la entrada del DUT. Como antes, los acopladores direccionales son ideales ya que son directivos, tienen bajas pérdidas y un alto aislamiento de la señal reflejada. De todas formas, debido a la dificultad de hacer acopladores direccionales de banda ancha, se utilizan puentes (*bridge*) en lugar de éstos.

Los puentes pueden trabajar hasta la continua (DC) pero tienen más pérdidas, traducándose en menos potencia de señal inyectada al DUT.

Uno de los parámetros más importantes de los acopladores direccionales es la **directividad**. La directividad mide la capacidad de separar señales que circulan en

sentido contrario a través del acoplador. Se puede considerar como un rango dinámico para la medida de señales reflejadas.

Desafortunadamente, la separación de señales nunca es perfecta. Por ejemplo, echémosle una hojeda al funcionamiento de un acoplador direccional de tres puertos. Idealmente, la señal incidente no debería aparecer en el puerto acoplado. Pero en realidad parte de la energía se fuga hacia el puerto acoplado debido a que tiene un aislamiento finito, y como es de esperar, este efecto nos introducirá un error en la medida de la señal reflejada que, con un buen calibrado, es posible erradicar casi totalmente.

$$\text{Directividad (dB)} = \text{Aislamiento (dB)} - \text{Factor Acople (dB)} - \text{Pérdidas (dB)} \quad (3.1)$$

El error de directividad es la principal razón por la que se ve un patrón de rizado en muchas medidas de las pérdidas de retorno (S_{11}) ya que, en los picos del rizado la directividad se suma en fase con la reflexión del DUT y en otros casos, la directividad puede anular la reflexión proveniente del DUT y generar un desvanecimiento muy pronunciado de las pérdidas.

○ Tipos de detectores

Hay dos maneras básicas de detectar las señales en los analizadores de redes: mediante diodos detectores o mediante receptores sintonizables.

Los **diodos detectores** convierten el nivel de señal de RF a un nivel proporcional de señal continua. Si la señal está modulada en amplitud, el diodo elimina la portadora de la modulación (o también llamado detección AC). El diodo es inherentemente escalar, así que se pierde la información de la portadora de RF.

Las dos principales ventajas de los diodos receptores es que cubren un amplio ancho de banda (desde 10MHz hasta 26.5GHz) y que son muy baratos comparados con los receptores sintonizables. Pueden medir señales de -60dBm o menos y tener un rango dinámico de 65 o 70dB dependiendo del tipo de detector. Su ancho de banda limita su sensibilidad y los hace vulnerables a los armónicos de la fuente y señales espurias.

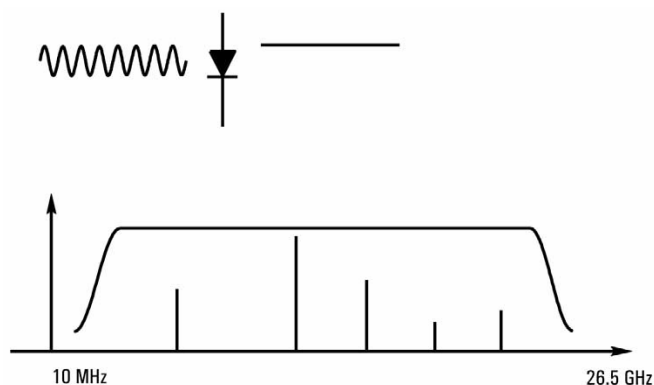


Fig. 31: Esquema funcional de los diodos detectores junto con su ancho de banda de trabajo.

Como los diodos detectores tienen un ancho de banda muy grande, a la técnica de detección con diodos también se le llama **broadband**.

Los **receptores sintonizables** utilizan un oscilador local (LO) para mezclar la señal de RF y bajarla a una frecuencia intermedia (IF). Para realizar el barrido en frecuencia de la señal entrante, se puede fijar el filtro de IF y variar la frecuencia del LO, o se puede fijar el LO y poner un filtro sintonizable en IF. Esta última configuración es la más utilizada por los analizadores de redes.

La señal de IF es filtrada con un filtro paso banda que estrecha el ancho de banda del receptor y mejora en gran medida la sensibilidad y el rango dinámico. Los analizadores modernos utilizan convertidores analógico-digital (ADC) y procesadores de señal digital (DSP) para extraer la información de la magnitud y la fase de la señal de IF.

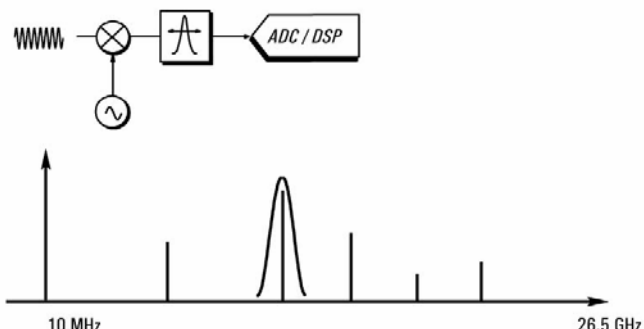


Fig. 32: Esquema funcional de los receptores sintonizables junto con su ancho de banda de trabajo.

Los receptores sintonizables ofrecen mejor sensibilidad y rango dinámico que los diodos detectores, y también ofrecen mejor rechazo a los armónicos y señales espurias. Además, el estrecho filtro de IF produce un descenso considerable del suelo de ruido (*noise floor*) dando como resultado una mejora considerable de la sensibilidad. Para mejorar la medida del rango dinámico podemos incrementar la potencia entrante, disminuir el ancho de banda del filtro de IF, o promediar.

Las dos últimas técnicas establecen un compromiso entre el *noise floor* y la velocidad de medida. Mediante el promedio se reduce el *noise floor* (cosa que no sucede con otros dispositivos como el analizador de espectros) porque el analizador de redes trabaja con números complejos. Sin información de la fase, el promediado no mejora la sensibilidad del analizador de redes.

Debido a la misma banda estrecha del filtro de IF, se incrementa el rango dinámico y se reducen los armónicos y señales espurias ya que al pasar la señal de RF por el mezclador del OL, los armónicos se desplazan a frecuencias fuera de la banda de paso del filtro de IF y son eliminados.

Y al igual que pasaba con los diodos detectores, a esta técnica se le llama **narrowband** ya que el filtro de IF es de banda estrecha.

3.1.3. Tipos de medidas

Una vez explicados los tipos de detectores que utilizan los analizadores de redes, es interesante saber que tipo de medidas pueden realizar y como las hacen. Básicamente hay dos tipos de medidas: el test transmisión/reflexión (o T/R) y los parámetros S .

En la **medida T/R**, la potencia de RF siempre sale del puerto 1 y el puerto 2 es el que se conecta al receptor del analizador. Para medir la señal que circula en sentido contrario, tenemos que desconectar el DUT, darle la vuelta intercambiando los puertos 1 y 2, y volverlo a conectar. La medida T/R sólo ofrece la respuesta de un puerto y es por eso que la precisión de la medida es menor si la comparamos con la medida de parámetros S . De todas formas, los analizadores con medida T/R son más económicos.

La **medida de parámetros S** permite la medida tanto de las señales incidentes como reflejadas en ambos puertos del DUT ya que son necesarias para la caracterización de los parámetros de la matriz S . La potencia de RF puede salir por cualquiera de los dos puertos de test y deben estar ambos conectados al receptor. La medida de parámetros S permite la corrección de errores en ambos puertos (12 parámetros) y es la forma más precisa y recomendable de hacer mediciones. Cabe decir que la medida de parámetros S ofrece mejor rendimiento en comparación con la medida T/R, pero los componentes de RF necesarios para la medida son más caros.

Para hacer el cambio de sentido del puerto 1 al puerto 2 de la señal inyectada en el DUT, se utilizan interruptores de transferencia (*transfer switch*). Hay dos tipos diferentes de interruptores que pueden usarse en la medida de parámetros S : los *solid-state* y los mecánicos.

Los interruptores *solid-state* tienen la ventaja de tener una vida útil infinita (siempre y cuando la potencia proveniente del DUT no sea excesiva). Como contrapartida, tienen unas pérdidas considerables que se traduce en una menor potencia inyectada al DUT para realizar las medidas.

Los interruptores mecánicos tienen menores pérdidas en comparación con los *solid-state* y son capaces de proporcionar una potencia de salida mayor. Su principal desventaja es que pueden empezar a desgastarse a partir de los 5 millones de ciclos de medida. Así, cuando se utilizan analizadores de redes con interruptores mecánicos, es habitual hacer las medidas en modo de barrido simple para no desgastar en exceso el interruptor mecánico.

Para terminar con este apartado, comentar que la medida de parámetros S puede utilizar interruptores de transferencia de 3 receptores o de 4 receptores y, dependiendo del tipo de interruptor, su calibración es diferente.

3.1.4. Errores en la medida

En la medida realizada por un analizador de redes existen tres tipos de errores: los errores sistemáticos, los errores de deriva y los errores aleatorios.

Los **errores sistemáticos** son debidos a las imperfecciones del analizador de redes y su puesta en funcionamiento. Son errores que se repiten en todas las medidas y por lo tanto se dice que son errores predecibles y se asumen como invariantes en el tiempo. Los errores sistemáticos se caracterizan en el proceso de calibrado y son eliminados de la medida de manera matemática por parte del procesador. Los errores de este tipo más comunes son la pérdida de señal a la entrada debido a la directividad del acoplador direccional, el *crosstalk*, las reflexiones en los conectores de fuente y carga, etc.

Los **errores aleatorios** son impredecibles ya que varían en el tiempo de una manera aleatoria y por lo tanto no pueden ser eliminados en el proceso de calibración. La mayor parte de los errores aleatorios son debidos a los ruidos de los componentes del analizador de redes, como pueden ser el ruido de fase de la fuente, el ruido de las muestras, el ruido de IF, etc.

Los **errores de deriva** son causados también por el analizador de redes pero en este caso al cambio de las condiciones de medida y rendimiento después de la calibración. Principalmente son causados por la variación de temperatura pero este efecto puede eliminarse realizando calibraciones entre las diferentes medidas. El rango de tiempo durante el cual la calibración se mantiene exacta depende del tiempo en que las condiciones de la medida supuestas por el usuario no varían. Si se crea un entorno en que la temperatura no varíe demasiado, normalmente se consigue un largo período de validez de la calibración a la par que se minimizan los errores de deriva.

3.1.5. Técnicas de corrección de errores y calibrado

Existen principalmente dos técnicas para corregir los errores en un analizador de redes: la normalización y la corrección mediante vectores.

En la **técnica de normalización** sólo hace falta un calibrado mediante la conexión directa de los puertos en donde se situará el DUT. El inconveniente de esta técnica es que sólo corrige algunos errores sistemáticos de los 12 posibles mediante otras técnicas. El principio de funcionamiento es el siguiente: se almacena en memoria una referencia de la medida y las siguientes medidas se dividen por la referencia almacenada. Existe una variante de esta técnica que consiste en hacer un calibrado en cortocircuito y otro en circuito abierto para posteriormente promediar las medidas con los valores obtenidos. Para la detección del nivel de señal se utilizan diodos detectores de banda ancha.

La **corrección mediante vectores** requiere de un analizador que pueda medir las magnitudes y las fases de las señales. Como es de esperar, este tipo de corrección de errores requiere un mayor

calibrado pero es capaz de corregir un mayor número de errores sistemáticos y conseguir una mayor precisión en la medida.

El principio de funcionamiento de la corrección de errores mediante vectores se basa en caracterizar los errores sistemáticos mediante la calibración del dispositivo con estándares conocidos y posteriormente ser capaces de eliminar estos errores de las medidas posteriores.

Dependiendo del número de puertos en que se realiza la calibración se diferencian dos métodos de corrección de errores mediante vectores: calibración de un puerto (*One-port calibration*) y calibración de dos puertos (*Two-port calibration*).

La **calibración de un puerto** se utiliza para las medidas de reflexión y puede eliminar tres tipos de errores sistemáticos (directividad, desequilibrio en la fuente y coeficiente de reflexión).

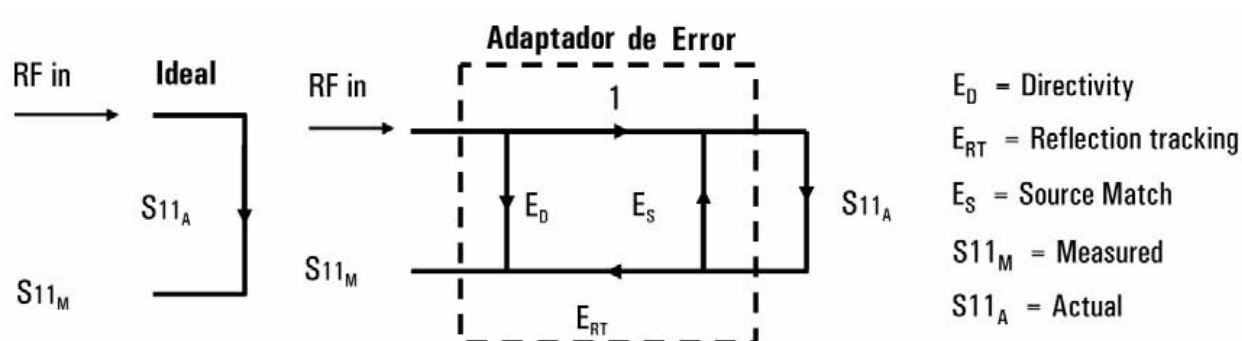


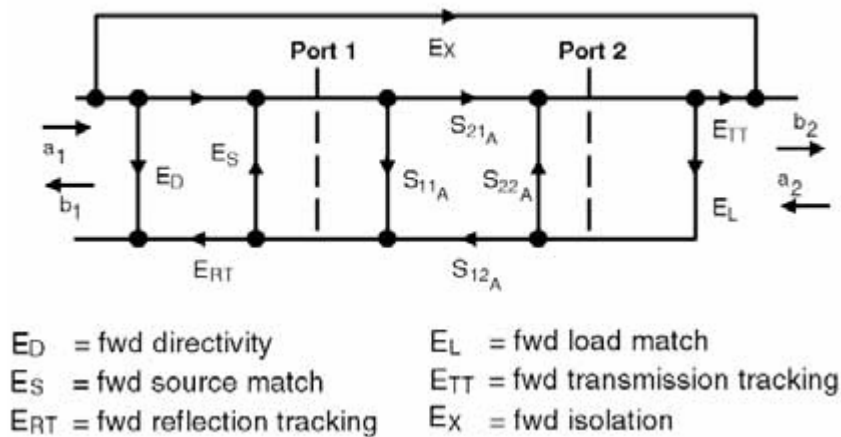
Fig. 33: Modelo de corrección de errores mediante la técnica de calibración de un puerto [19].

En el caso de la Fig. 33, se tiene que en la medida de las pérdidas de retorno de un puerto hay tres errores sistemáticos (E_D , E_{RT} y E_S) y el parámetro S_{11M} a resolver. Para resolverlo, primero se tienen que calcular todos los términos de error de manera individual creando tres ecuaciones más con una incógnita cada una y resolviéndolas simultáneamente. Las tres ecuaciones salen de medir tres estándares de calibración conocidos (por ejemplo cortocircuito, circuito abierto y carga adaptada). Una vez resueltas estas tres ecuaciones, es posible calcular las pérdidas de retorno S_{11M} del DUT.

Si se realizan medidas de circuitos con dos accesos mediante la técnica de corrección de errores con calibración de un puerto, se asume que el segundo puerto está bien terminado, es decir, con una carga adaptada estándar. Si por el contrario el segundo puerto se conecta al analizador de redes y el aislamiento del DUT a la señal reflejada es bajo (por ejemplo un filtro paso banda o unos cables), ya no se cumple la condición de buena terminación del segundo puerto. En estos casos, la técnica de calibrado de dos puertos ofrece resultados más precisos. Un ejemplo contrario al anterior en que no influye esta calibración es en un amplificador ya que el aislamiento del amplificador permite la calibración de sólo un puerto.

La **calibración de dos puertos** se utiliza para las medidas de reflexión y de transmisión y es capaz de eliminar doce tipos de errores sistemáticos. Es por eso que requiere un calibrado de doce parámetros mediante estándares conocidos (cortocircuito, circuito abierto, carga adaptada, y conexión directa) aunque algunos de los estándares son medidos varias veces, como por ejemplo la conexión directa que se mide cuatro veces.

Modelo de transmisión



Modelo de reflexión

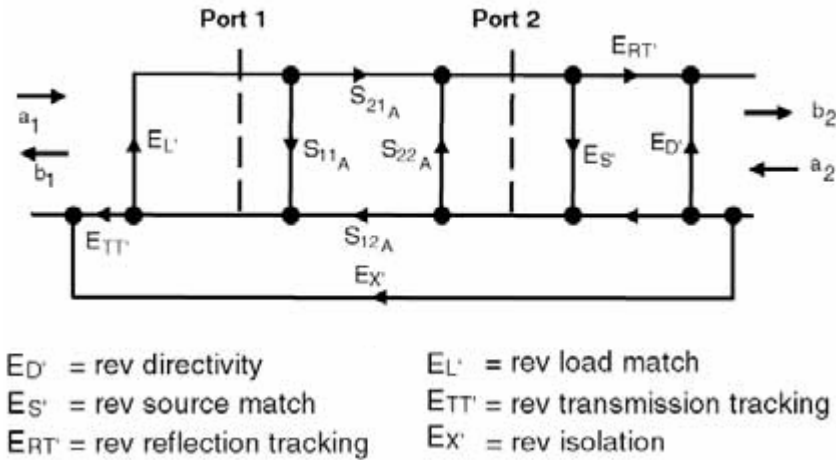


Fig. 34: Modelo de corrección de errores mediante la técnica de calibración de dos puertos [19].

La calibración de dos puertos es la forma más precisa de corregir errores sistemáticos ya que se tienen en cuenta las principales fuentes de errores. Como puede verse en la *Fig. 34* hay cuatro incógnitas a resolver coincidentes con los cuatro parámetros S del DUT (S_{11A} , S_{12A} , S_{21A} , S_{22A}). Para su cálculo hay que resolver primero los doce términos de error que, como en el caso de un puerto, se obtienen mediante el calibrado con estándares conocidos. Hay que tener en cuenta que cada uno de los parámetros S depende de los otros, y es debido a esto que el procesador del analizador debe ir resolviendo las ecuaciones por pasos.

Cabe destacar que los estándares de calibración están definidos en la hoja de calibrado que se encuentra almacenada en el analizador y debe asegurarse que el kit de calibrado corresponde con el estándar utilizado.

Para terminar se comparan las incertidumbres de los diferentes tipos de corrección de errores sin entrar en excesivos detalles.

La diferencia entre la incertidumbre en la medida con calibración de dos puertos y las otras técnicas es muy favorable a la primera. En el caso de reflexión, la incertidumbre de la calibración de dos puertos puede llegar a ser diez veces menor que la calibración de un puerto. Bien es cierto que si utilizamos un atenuador a la hora de hacer las medidas mejora sensiblemente la incertidumbre de la técnica de un puerto.

Si comparamos la técnica de un puerto, la técnica de dos puertos y la normalización, se observa que la normalización es la que peor incertidumbre proporciona.

Por ejemplo, en la medida de un filtro paso bajo con 1dB de pérdidas de inserción en la banda de paso y 16dB de pérdidas de retorno, con la calibración de un puerto se puede obtener una incertidumbre para reflexión de $-4,6 / +10,4\text{dB}$. Sin embargo para la misma medida con la calibración de dos puertos se obtiene una incertidumbre de $-0,44 / +0,53\text{dB}$.

E idénticamente pasa si comparamos la medida de transmisión con la técnica de normalización, que proporciona una incertidumbre de $-0,8 / +0,82\text{dB}$, mientras que la calibración de dos puertos consigue una incertidumbre de $\pm 0,05\text{dB}$.

Como conclusión decir que para medir los parámetros S de un circuito de dos accesos es mejor realizar una corrección de errores vectorial con calibración de dos puertos para obtener resultados fiables y con la mayor precisión posible. Y este ha sido el caso de este proyecto. Cada vez que se han realizado medidas se ha calibrado el analizador de redes por los dos puertos, con 4 tipos de estándares (cortocircuito, circuito abierto, carga adaptada y conexión directa).

4. Implementación del proyecto

En este apartado se documenta todo el proceso de realización del proyecto. Se divide en cuatro apartados principales: hardware, software, firmware y estructura de soporte. En el apartado hardware se explica la implementación de la placa de circuito impreso, sus funcionalidades, sus componentes y su conexión. En el siguiente apartado, el software, se documenta todo el programa de control, la obtención de datos por parte del PC y la comunicación con el hardware. En el apartado dedicado al firmware se explica el software programado en el microcontrolador que controla las fases del motor según las ordenes del PC. Y en el último apartado, el de estructura de soporte, se explica cómo se ha fabricado el conjunto que sostiene el motor, el hardware y el filtro.

4.1. Inicios del proyecto

Desde el principio, las funcionalidades del proyecto estaban muy claras: diseñar un sistema que controlara un motor a pasos que, actuando sobre el tornillo de sintonización de un filtro de microondas, fuera capaz de centrarlo a la frecuencia deseada por el usuario automáticamente.

El punto de partida fueron dos funciones programadas en Matlab que, lo único que hacían, era leer los parámetros S medidos en el analizador de redes. Estas dos funciones son *enviar* y *rebre* que más adelante, en el apartado *Software*, se explicarán con más detalle. A partir de aquí, se decidió hacer el programa de control en entorno Matlab para que el tratamiento de los datos fuera más sencillo y poder trabajar sin problemas con números complejos, logaritmos y poder configurar puertos de manera ‘sencilla’. Además, Matlab ofrece la posibilidad de representar muy fácilmente los resultados de manera gráfica.

La primera opción que se barajó fue configurar el puerto paralelo del PC con cuatro salidas, una por fase, que actuarán directamente sobre las cuatro fases del motor. Rápidamente se desestimó ya que el puerto paralelo no puede entregar toda la corriente necesaria para mover el rotor y además, el puerto paralelo trabaja con niveles de tensión positivos y negativos de 15V.

Posteriormente y basándose en la primera opción, se planteó poner unos transistores de potencia en las cuatro señales provenientes del puerto paralelo, previa transformación a niveles TTL, para atacar las cuatro fases del motor.

Leyendo información sobre motores a pasos y demás, se observó que nunca debe conectarse directamente un motor a un hardware, sino que debe aislarse mediante optoacopladores la etapa de maniobra y la etapa de potencia. También se vio que para evitar daños al motor y no actuar sobre fases opuestas, existen unos integrados (tipo L293, etc.) que, diciéndole la dirección de

giro y el tipo de paso (entero o medio), actúa sobre las fases para girar el rotor. Estos integrados son muy interesantes pero muy caros y se averían con facilidad debido a la gran cantidad de corriente que circula por ellos, unos 4A.

Así que finalmente, se decidió poner un microcontrolador en el hardware para dotarlo de autonomía y poder comunicarse de manera fácil con el PC mediante puerto serie. Además, se facilita la gestión de errores y se garantiza un sistema muy versátil y adaptable. En lugar de poner un integrado tipo L293 se decidió poner unos transistores de potencia de gran capacidad (1A) aislados por unos optoacopladores de la etapa lógica, y todo ello controlado por el microcontrolador.

Una vez decidido el diseño del hardware, se trabajó en el software de control que gestiona todo el proceso de sintonización. Como se ha dicho anteriormente, se ha diseñado en un entorno Matlab y se compone de diferentes funciones gestionadas por un *main*.

Paralelamente se buscó un microcontrolador apropiado a las necesidades del proyecto y se inició el diseño del firmware que se comunica con el software de control mediante el puerto serie. En este punto se establecieron las bases del protocolo de comunicación entre el software de control y el firmware del microcontrolador en función de las acciones a realizar por el hardware sobre el motor a pasos.

4.2. Hardware

El hardware hace las funciones de comunicación con el PC y junto con el microcontrolador, activa las señales necesarias para actuar sobre el motor, gestiona los errores de la placa y genera las tensiones necesarias para los diferentes dispositivos.

El trabajo comienza separando todo el hardware en etapas más pequeñas para trabajar más cómodamente e ir testeando los diferentes dispositivos. Todo ello se monta en una placa protoboard y se verifica el correcto funcionamiento y el consumo de corriente. Una vez probada cada parte, se unen entre si y se vuelve a comprobar el funcionamiento global.

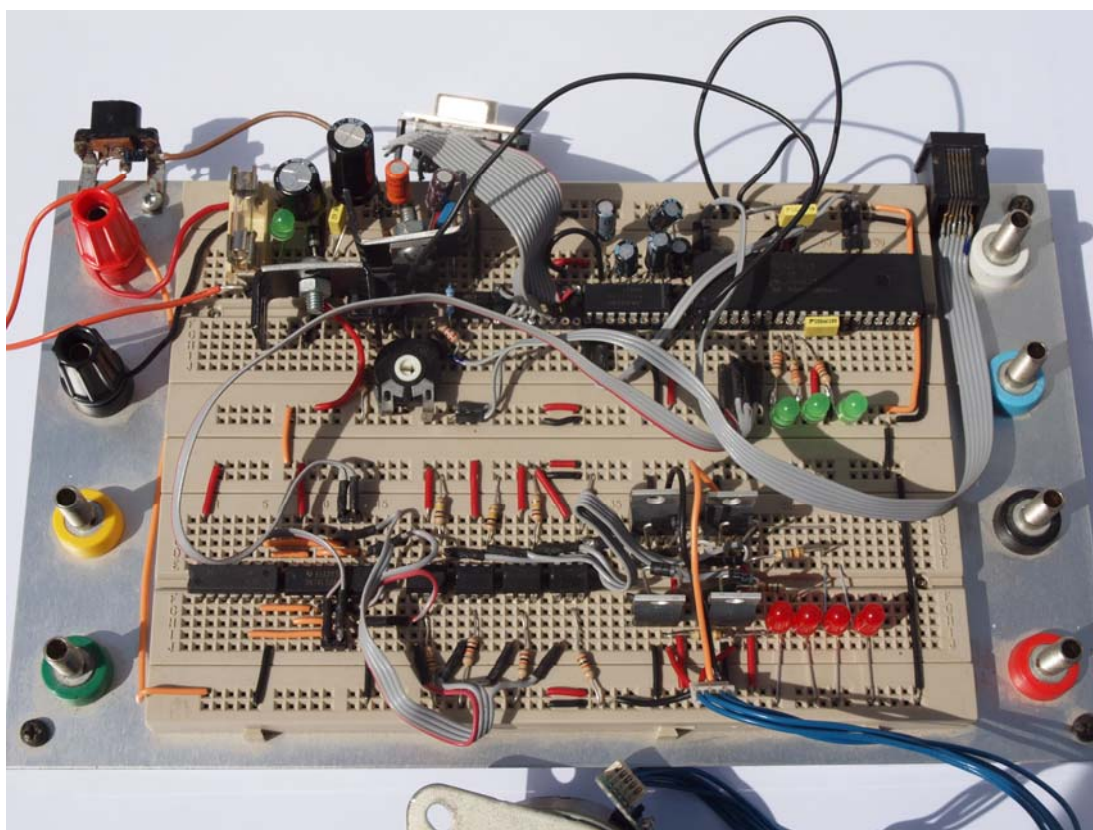


Fig. 35: Protoboard del hardware del proyecto.

Cuando todo funciona correctamente en la protoboard, se pasa al diseño digital mediante el programa *Altium Designer 6.6* para hacer el circuito impreso en una sola placa o PCB (del inglés *Printed Circuit Board*).

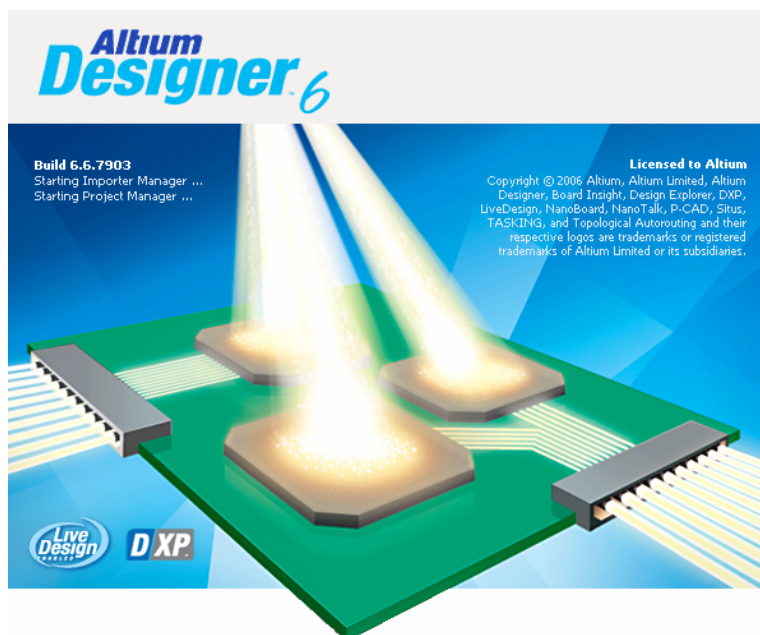


Fig. 36: Inicio del programa Altium Designer 6.

El programa *Altium Designer 6.6* es un extenso software para el diseño de circuitos PCB, diseños FPGA y desarrollo de software *embedded*. Es uno de los más utilizados en la industria por su potencia de trabajo, versatilidad y capacidad de adaptación en cada proyecto. Dispone de librerías estándares con infinidad de componentes de diferentes fabricantes. Cada componente está parametrizado para realizar simulaciones de los circuitos antes de su fabricación. Además, crea una imagen de la PCB para hacerse una idea de cómo estarán dispuestos los dispositivos y poder juntarlos para minimizar el espacio.

Cabe destacar que nunca se había utilizado este programa ni hecho el diseño de circuitos integrados, pero gracias a los diferentes tutoriales y *papers* de ayuda, se ha terminado haciendo un buen trabajo, aunque seguramente, mejorable.

4.2.1. Funcionalidad del hardware y diagrama de bloques

El hardware tiene cuatro funciones principales:

1. Comunicarse con el software de control del PC mediante el puerto serie.
2. Activar las fases del motor a pasos según las órdenes del software de control.
3. Gestionar las señales de los pulsadores de paro o paso del mismo hardware.
4. Habilitar un entorno para programar el firmware del microcontrolador.

Para realizar estas cuatro funciones se ha dividido el hardware en bloques más pequeños cuyo diseño modular facilitará la implementación de todo el conjunto. Por esta razón se ha dividido en

siete partes y su diseño esquemático se ha realizado de manera independiente pero teniendo en cuenta las salidas y entradas en la interconexión de los diferentes módulos.

Primeramente se diseñan los esquemáticos de cada módulo (*.SchDoc en la Fig. 37). Una vez terminados, se crea un objeto de cada bloque que tiene solamente las entradas y salidas del esquemático para poder conectarlo con otros módulos (U_* en la Fig. 37). De esta forma nos aseguramos no crear esquemáticos demasiado grandes.

Seguidamente se crea otro esquemático que contiene las conexiones entre los diferentes módulos. A este esquemático con los diferentes bloques se le llama *TOP*, ya que de él cuelgan los diferentes módulos. De esta forma se crea una estructura jerárquica sabiendo en todo momento y de forma simple que bloques pertenecen a ese *TOP*. Esta estructura jerárquica puede ir incrementándose tantos niveles como sea necesario.

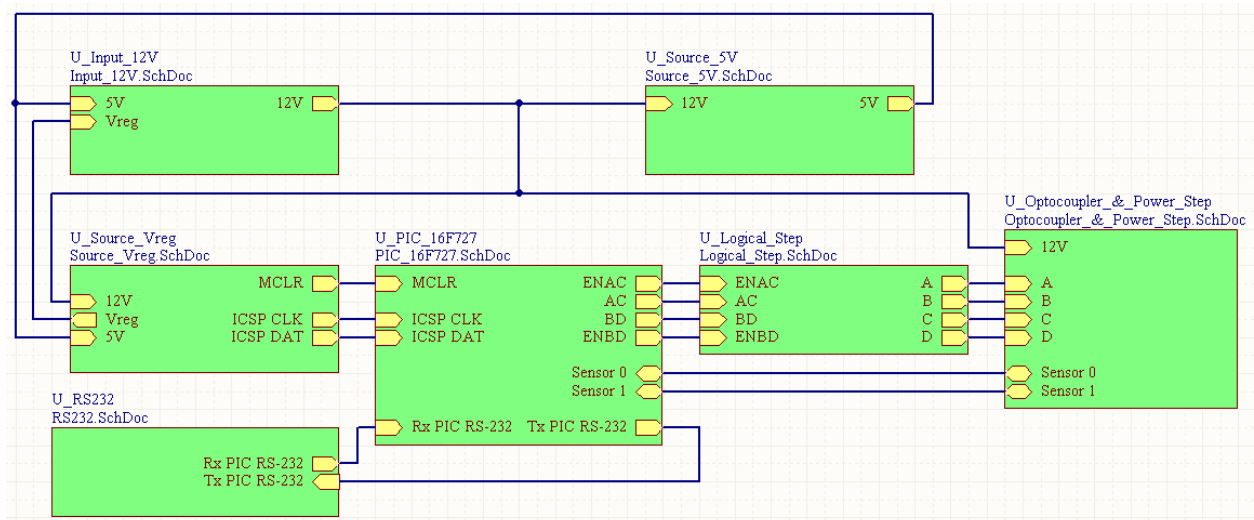


Fig. 37: Archivo *TOP_PCB.SchDoc*. Diagrama e interconexión de los bloques componentes del hardware.

Como puede verse en la Fig. 37 hay siete bloques conectados entre si y estas mismas conexiones sirven posteriormente para hacer el *routing* del PCB. Cada módulo tiene una funcionalidad específica con sus entradas y salidas de/hacia otros módulos. Los conectores de voltaje, jacks, y salidas hacia el motor no se reflejan en este esquema ya que están contemplados dentro de los esquemáticos como conectores y no forman parte de la conexión entre módulos, sino que son entradas o salidas hacia el exterior del hardware.

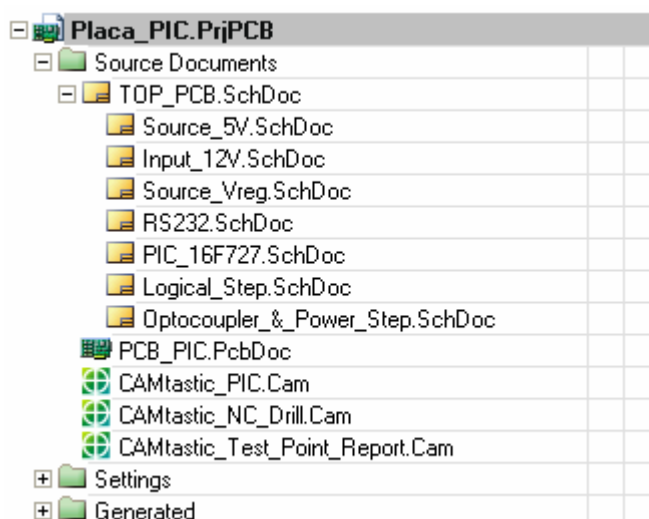


Fig. 38: Jerarquía de los directorios del proyecto.

En la Fig. 38 puede verse la jerarquía de los documentos del proyecto. En el primer nivel se encuentra el proyecto *Placa_PIC.PrjPCB*, que se trata del *workspace* con las diferentes características del proyecto (tipo PCB, reglas de *routing*, esquemáticos, *footprints* utilizados, características, etc.). Dentro de él se encuentran las diferentes carpetas de archivos: *Source Documents* para los archivos fuente como esquemáticos, software y demás, *Settings* en donde se ubican los *Output Job Files* con los *reports*, BOMs, *prints*, etc. y en *Generated* se guardan los archivos generados para la fabricación del PCB como *layouts*, *drills*, *mechanical layers* y otros.

En la carpeta *Source Documents* está ubicado el archivo *TOP_PCB.SchDoc* que es el archivo con los diferentes módulos representados como cajas con sus entradas y salidas que se interconectan entre sí para crear el esquema eléctrico de todo el hardware (ver Fig. 37). Y de éste *TOP* cuelgan los diferentes esquemáticos que lo conforman ya con todos los componentes e integrados.

Dentro de la carpeta anterior también se encuentra el archivo *PCB_PIC.PcbDoc*, que es la representación gráfica del PCB finalizado con la disposición física de los componentes y las pistas. Este documento es muy importante ya que es el PCB que se quiere crear y es en donde deben ubicarse correctamente los diferentes elementos para cumplir con las dimensiones físicas y realizar el *routing* de las pistas. Los archivos *CAMtastic_** son generados a posteriori para poder exportar el PCB a otras aplicaciones, crear documentos como el BOM o generar el archivo de agujeros para el agujereado de la placa mediante una fresadora CNC (*Computed Numerically Controlled*)

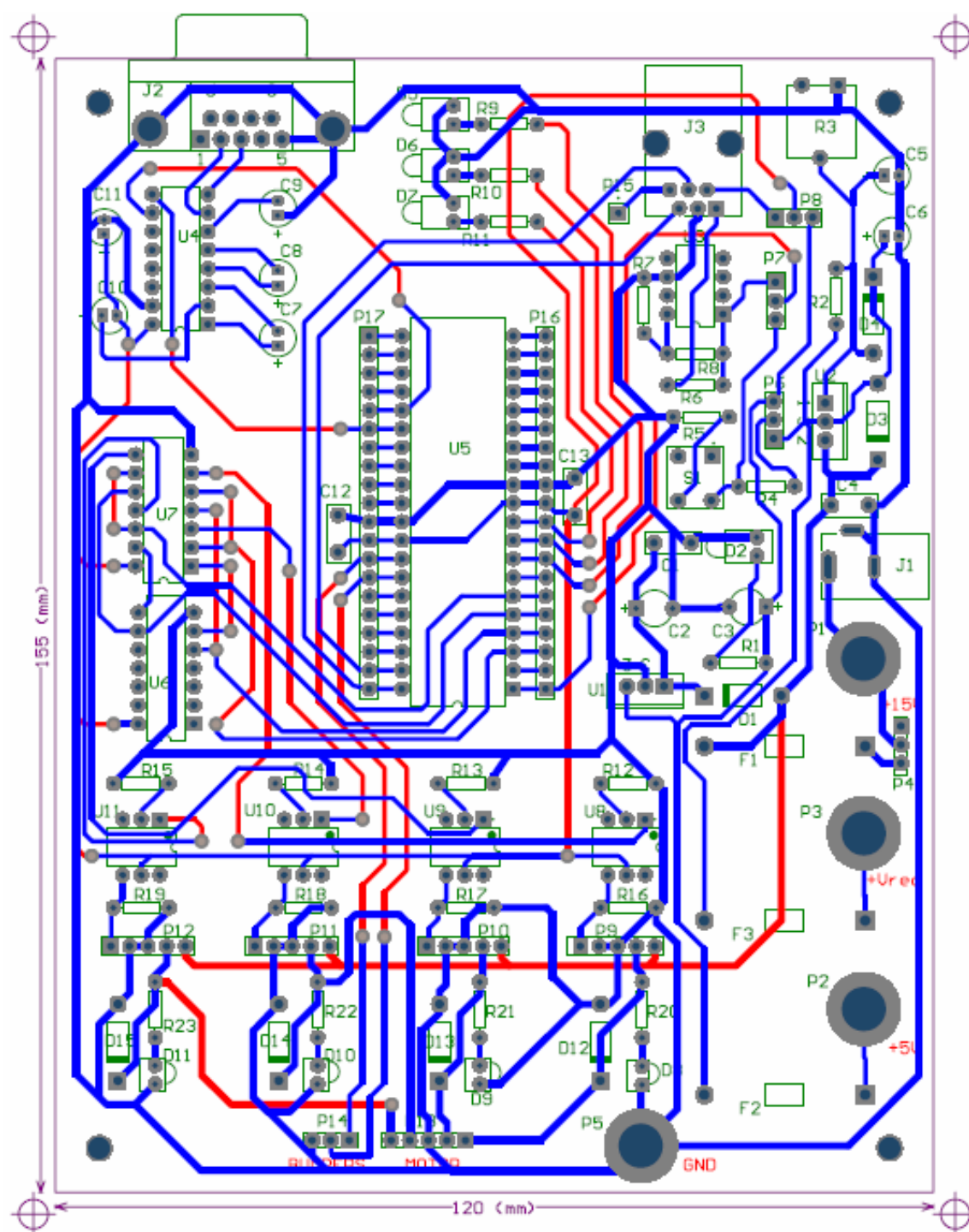


Fig. 39: Archivo *PCB_PIC.PcbDoc*. Vista general del hardware con los dispositivos y las pistas.

En la Fig. 39 se puede ver una imagen global del PCB creado para este proyecto. Se muestra el contorno y dimensiones de la placa con las cruces de alineación de los dos *layouts*, los agujeros de los dispositivos y sus *footprints*, las pistas de la cara superior (en rojo) y las de la cara inferior (en azul).

El *routing* de las pistas ha sido mixto (automático y manual) ya que en un principio se realizó sólo por la cara inferior. Pero debido a la gran complejidad del circuito y a la posibilidad de

realizarlo por las dos caras (superior e inferior), se compactaron más los componentes y se pudo disminuir el tamaño del circuito. Después de realizar el *auto-routing* disponible en el programa, se corrigieron todas las pistas a mano y se modificaron muchas de ellas para que el conjunto fuera más simétrico y evitar distancias entre pistas demasiado pequeñas.

Una vez creado el PCB, se imprimió una imagen con la posición de los componentes y sus agujeros. Luego se verificó que realmente no había solapamiento entre los dispositivos y que todo fuera correcto. Se ajustaron las posiciones de algunos componentes y se dio el visto bueno a los *layouts*.

Nótese que para el *auto-routing* existen unas “reglas” que deben cumplirse, como por ejemplo el espacio entre pistas, el ancho de las pistas, el ángulo máximo de giro de las pistas, etc. pero previamente tienen que crearse estas reglas. Como reglas especiales destacan el ancho de las pistas de GND y 12V, que son de 1mm mientras que las otras son estándar de 0,6mm.

En los siguientes apartados se presenta cada uno de los siete módulos que constituyen el hardware, indicando sus funcionalidades principales así como el esquema eléctrico diseñado.

4.2.2. Entradas y salidas de tensión

En un funcionamiento normal del hardware hay una única entrada de tensión de 12V DC y dos salidas auxiliares de tensión: una a 5V DC y otra a una tensión regulable Vreg DC. También hay la conexión de referencia o tierra (GND).

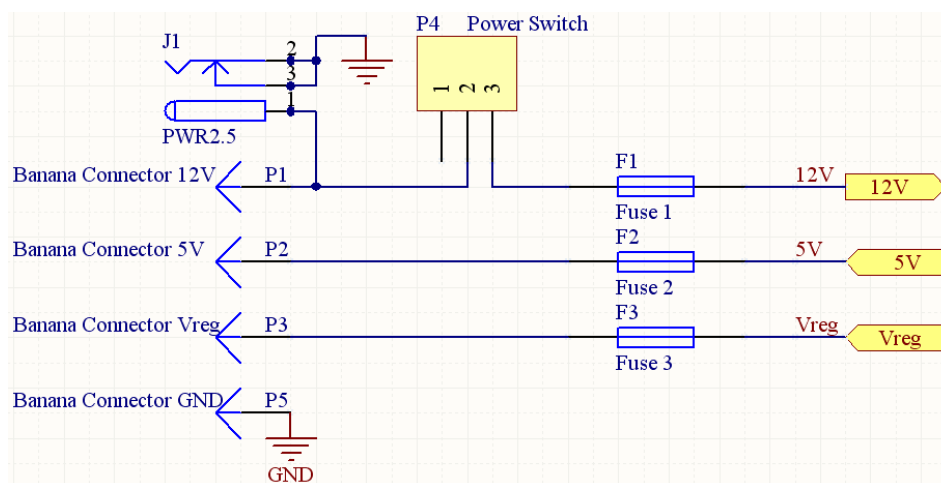


Fig. 40: Esquema eléctrico de las entradas y salidas de tensión.

Como puede verse en la Fig. 40, la entrada de 12V DC puede realizarse mediante conector tipo banana o mediante conector estándar de 2.5mm (los típicos de conexiones DC en cargadores, juguetes, etc.). Así se garantiza una versatilidad del hardware adaptable a todo tipo de escenarios aunque no se disponga de un generador de tensión regulable a 12V DC. Tras los conectores de entrada a 12V DC, se ha situado un interruptor general que puede cortar la tensión en todo el

hardware. A continuación del interruptor se ha intercalado un fusible de protección de 1.6A para evitar sobrecargas de corriente.

El conector de 5V DC tipo banana debe utilizarse normalmente como salida auxiliar. Este conector se ha puesto por si hubiera sido necesario ubicar un ventilador para refrigerar los transistores de potencia, o para futuras placas de expansión. Cabe destacar que este puerto puede trabajar como entrada si no disponemos de una alimentación a 12V DC. Un claro ejemplo de esta configuración es si se desea programar el microcontrolador o comprobar la maniobra sin conectar el motor. A esta salida también se le ha insertado un fusible de 1.5A, que es la máxima corriente que puede entregar el regulador de tensión de 5V DC.

El conector de Vreg DC tipo banana es una salida de tensión regulable entre 1.2V y 11.5V DC. La decisión de poner una fuente de tensión regulable fue tomada al descubrir que muchos microcontroladores necesitan una tensión de reset superior a 5V DC (por ejemplo entre 8V-9V DC) y para que el hardware fuera versátil en la elección del microcontrolador, se decidió poder elegir la tensión de reset entre 5V DC o Vreg DC. Y ya que se ha ubicado una fuente de tensión regulable, se ha puesto un conector de salida auxiliar a esta tensión por si fuera útil en un futuro. Al igual que el esquema de 5V DC, también se ha ubicado un fusible de 1.5A para limitar la corriente entregada por el regulador de tensión.

Y finalmente un conector de tierra tipo banana para conectar la referencia de tensión cuando se alimenta el hardware con los conectores tipo banana de un generador de tensión, o para obtener fácilmente la referencia al hacer medidas con el multímetro digital.

4.2.3. Fuente de 5V DC

Para alimentar todos los integrados y el microcontrolador, se ha incorporado en el hardware un circuito regulador de tensión de 5V DC. Como los bobinados del motor trabajan a 12V DC, se ha escogido la opción de poner solamente un conector de entrada de tensión a 12V y ya dentro del hardware hacer la conversión a 5V DC para alimentar el resto de integrados y componentes. Así sólo hace falta una conexión a una fuente de alimentación externa.

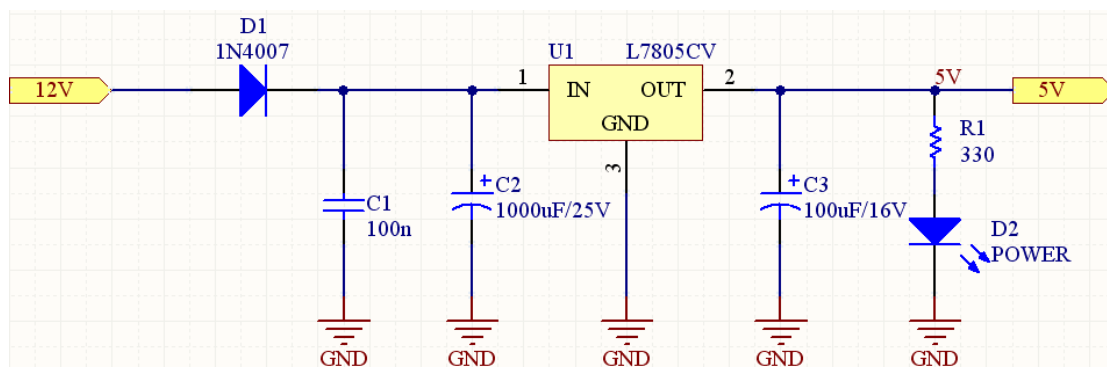


Fig. 41: Esquema eléctrico de la fuente de 5V DC.

Como regulador de tensión se ha escogido un L7805 ya que se había utilizado en alguna ocasión y se estaba familiarizado con sus principales características. Además de ofrecer muy buena estabilidad de la tensión puede entregar hasta 1.5A de corriente.

Como indicador del funcionamiento de la alimentación del hardware, se ha ubicado un diodo led verde que se enciende cuando hay alimentación de 5V DC, y normalmente como los 5V DC están generados por el regulador de tensión alimentado con 12V DC, da testigo de la conexión a 12V DC del hardware.

4.2.4. Fuente de Vreg DC, MCLR e ICSP

Como ya se ha comentado anteriormente, se ha decidido insertar una fuente de tensión regulable ya que muchos microcontroladores requieren una tensión de reset superior a los 5V DC (normalmente entre 8V-9V DC). De esta manera, se puede escoger que tipo de tensión de reset se quiere: 5V DC, Vreg DC o la tensión proveniente del MPLAB ICD 2 (cuando se programa el microcontrolador o se conecta al PC).

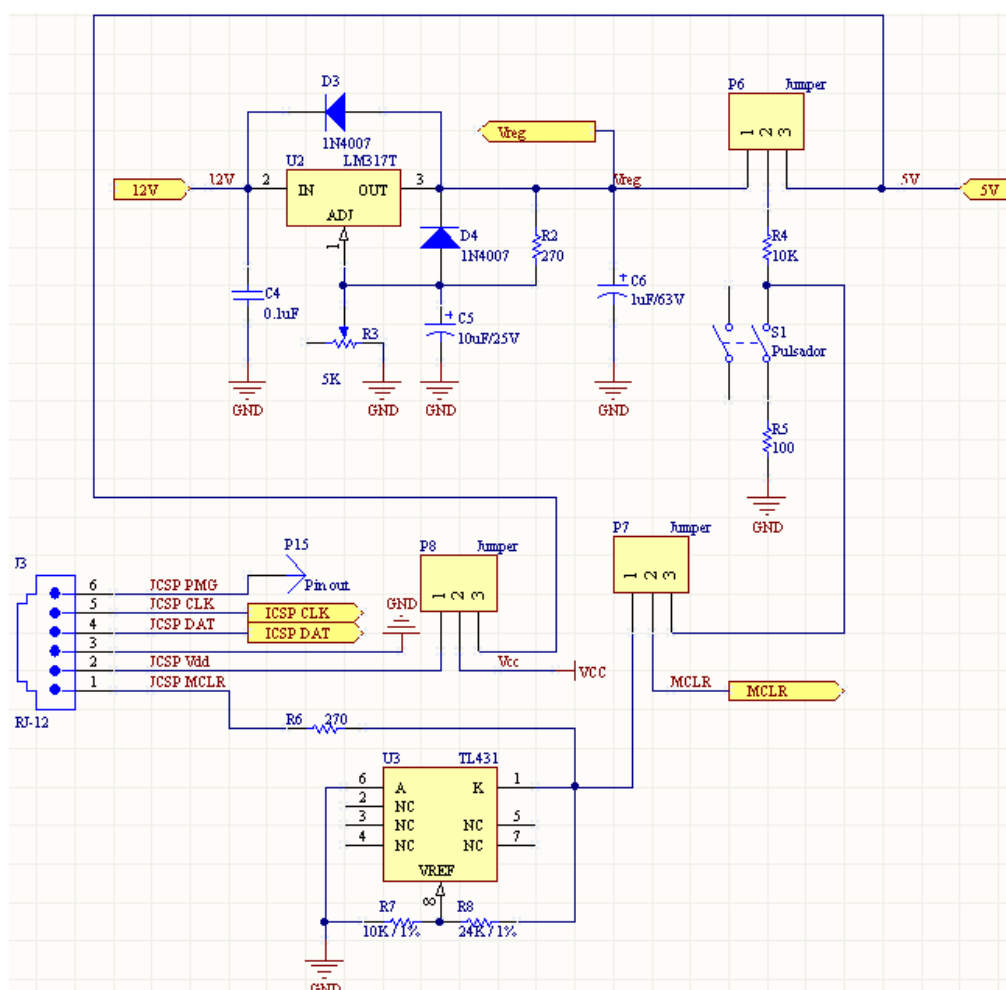


Fig. 42: Esquema eléctrico del regulador de tensión, tensión de reset y circuito auxiliar para la conexión del microcontrolador al MPLAB ICD 2.

En el esquema eléctrico de la Fig. 42 se puede dividir en tres partes: la fuente de tensión regulable, el circuito de reset y el circuito auxiliar para la conexión del microcontrolador con el MPLAB ICD 2.

En la parte superior izquierda de la imagen se encuentra ubicado el esquema del regulador de tensión. Se trata de una fuente regulable mediante un potenciómetro de 5K realizado con el integrado LM317. Este integrado es de gran versatilidad y ofrece unas características de estabilidad muy buenas, además de poder ofrecer hasta 1.5A de corriente. En condiciones normales de 12V DC de alimentación, ofrece un rango de tensión regulable entre 1.2V y 11.5V

A su derecha se encuentra el *jumper* 6 que permite elegir la tensión para el reset del microcontrolador entre 5V o Vreg. Del conector central del *jumper* 6 sale la conexión que ataca el *switch* de reset normalmente abierto (N.A.) y que al pulsarse, hace caer la tensión a la entrada reset del microcontrolador y realizar el reset.

Pero también es posible elegir mediante el *jumper* 7 la tensión de reset proveniente del conector RJ-12 que se conecta con el MPLAB ICD 2 previa regulación de la tensión mediante el integrado TL431.

Así existen tres tensiones de reset posibles: 5V, Vreg y la proveniente del MPLAB ICD 2 tal y como se muestra en la siguiente tabla.

Tabla 1: Tensiones de reset del microcontrolador.

TERMINALES JUMPER 6 (P6)	TERMINALES JUMPER 7 (P7)	TENSIÓN DE RESET
X	1-2	MPLAB ICD 2
1-2	2-3	Vreg
2-3	2-3	5V

En la parte inferior se encuentra el conector RJ-12 que se utiliza para conectar el microcontrolador con el dispositivo MPLAB ICD 2 de *Microchip*. Este dispositivo es el encargado de comunicar el microcontrolador con el PC mediante cable USB y se utiliza para programar el microcontrolador, debugar el programa y poder controlar el microcontrolador desde el PC. Con el microcontrolador empleado en este proyecto sólo se utilizan 5 señales digitales de las 6 disponibles en el conector.

Las 5 señales empleadas son: tensión de alimentación (Vdd), referencia (GND), tensión de reset del microcontrolador (MCLR), una señal de reloj para sincronizar el microcontrolador (CLK) y una señal de datos (DAT). Por si fuera necesario cambiar el microcontrolador, se ha dejado un pin con la señal (PMG) no utilizada del conector RJ-12. Así, con un simple cableado sería posible llevar esta señal al pin correspondiente del microcontrolador.

Como el microcontrolador se puede alimentar desde el MPLAB ICD2 mediante la señal Vdd, se ha insertado el *jumper* 8 para decidir si se alimenta el microcontrolador con los 5V de la fuente del propio hardware o si se escoge la alimentación proveniente del MPLAB ICD 2. Cabe destacar que cuando se conecta el microcontrolador al PC mediante el MPLAB ICD 2 alimentado por los 5V del hardware, aparece un error en el programa de control del microcontrolador. Esto es debido a que no detecta tensión en el pin de alimentación y piensa que el microcontrolador no está alimentado. Para solucionar esto, se ha construido un conector triple que conmuta todos los pins del *jumper* 8. Así, se consigue alimentar el microcontrolador mediante los 5V del hardware y que el MPLAB ICD 2 los detecte.

Tabla 2: Tensión de alimentación del microcontrolador.

TERMINALES JUMPER 8 (P8)	TENSIÓN ALIMENTACIÓN MICROCONTROLADOR
1-2	MPLAB ICD 2
2-3	5V
1-2-3	5V (sólo sin alimentación del MPLAB ICD 2)

4.2.5. Comunicación RS-232

Para comunicar el hardware con el PC, se ha utilizado el estándar RS-232. Este tipo de comunicación, en su forma más básica, tiene dos canales de comunicación: transmisión y recepción. El canal de transmisión del microcontrolador está conectado al canal de recepción del PC y viceversa. De esta forma no hay colisiones y los datos se almacenan en los *buffers* de entrada de cada dispositivo para ser leídos.

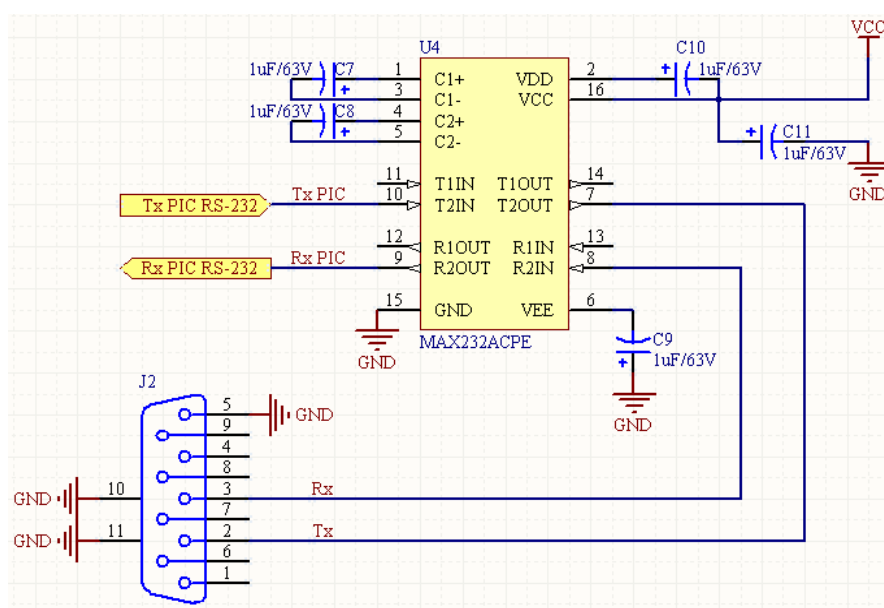


Fig. 43: Esquema eléctrico del bloque RS-232 en el hardware.

Como se puede ver en la Fig. 43 el conector J2 es un conector estándar DB-9 hembra. De él salen las señales de recepción (Rx) y transmisión (Tx) hacia el microcontrolador pasando por el MAX232 que adapta los niveles de señal del puerto serie del PC (+15V, -15V) a niveles TTL (0V, 5V).

Destacar que como no se hace control de flujo por hardware, no es necesario conectar ningún otro pin del conector DB-9 y que se han elegido los pins del MAX232 que ofrecen el *routing* más sencillo en el diseño del PCB.

4.2.6. Microchip PIC 16F727

Para dotar al hardware de autonomía se ha elegido un microcontrolador ya que en él se integran la CPU, la memoria de programa, la memoria RAM y los pins de I/O y programación. El modelo elegido ha sido el PIC 16F727 de *Microchip*. La elección del microcontrolador se ha realizado teniendo en cuenta las necesidades del proyecto y mediante una aplicación disponible en la página web de *Microchip* en la que, guiándose por una serie de pasos y dependiendo de la aplicación que se le dé, acaba proponiendo los microcontroladores más adecuados a nuestro proyecto.

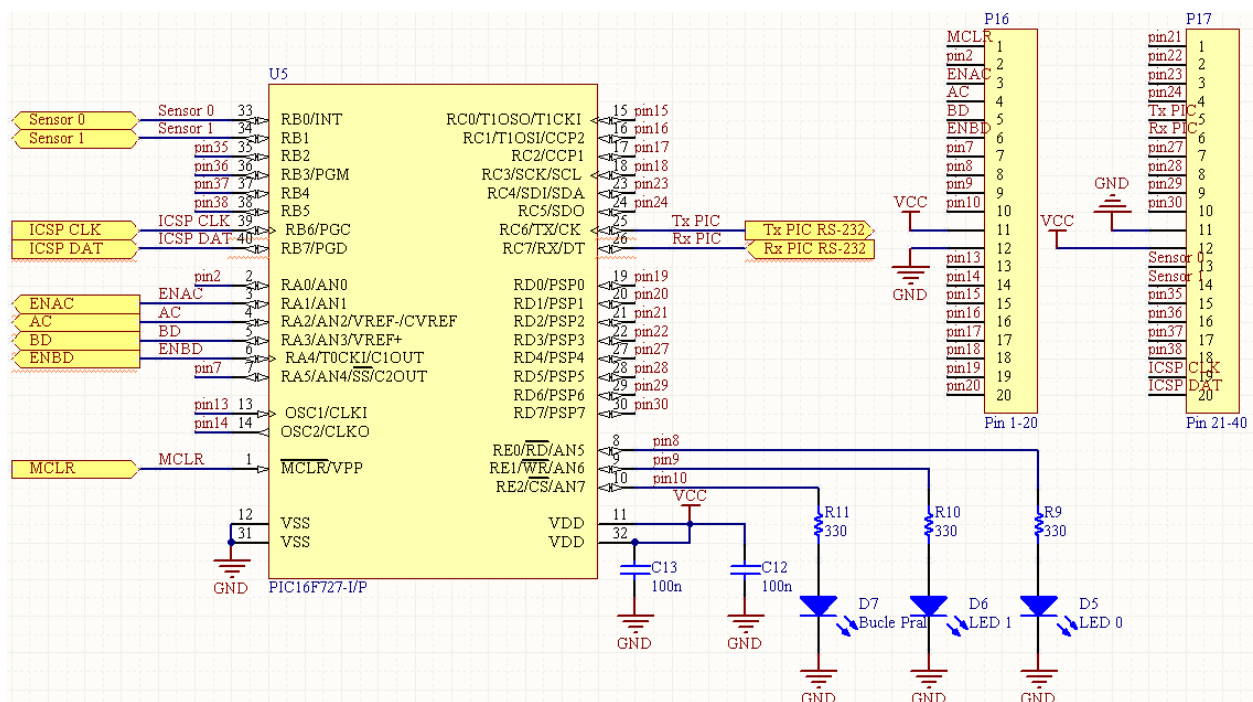


Fig. 44: Esquema eléctrico del microcontrolador con los leds de control y los pins de expansión.

En la Fig. 44 se observa el esquema eléctrico del bloque microcontrolador (U5) con los leds de control utilizados (D5, D6 y D7) y los conectores de expansión (P16 y P17) para testear tensiones o conectar futuras expansiones. Físicamente los conectores de expansión se han situado

al lado de los pins del microcontrolador para que sea más fácil su identificación. Los condensadores C12 y C13 se ubican en las entradas de alimentación del microcontrolador lo más próximos a él posible y sirven para estabilizar el voltaje y evitar desvanecimientos en la alimentación.

En la siguiente tabla se muestran las entradas del microcontrolador:

Tabla 3: Entradas al microcontrolador.

Entradas al microcontrolador	Función
Sensor 0 Sensor 1	Pins que al cambiar su tensión salta una interrupción. Utilizados como interruptores en el hardware.
ICSP CLK	Entrada de reloj al microcontrolador cuando se conecta al programador MPLAB ICD 2.
ICSP DAT	Entrada de los datos al microcontrolador cuando se conecta al programador MPLAB ICD 2.
MCLR	Entrada de reset del microcontrolador. Puede provenir del hardware o del programador MPLAB ICD 2.
Rx PIC RS-232	Entrada de datos del puerto RS-232 proveniente del PC.
Vcc	Tensión de alimentación a 5V DC.
GND	Tensión de referencia o tierra.

Y en la siguiente tabla las salidas del microcontrolador:

Tabla 4: Salidas del microcontrolador.

Salidas del microcontrolador	Función
ENAC	Señal que activa las fases A y C del motor.
AC	Fase A o C del motor.
BD	Fase B o D del motor.
ENBD	Señal que activa las fases B y D del motor.
Tx PIC RS-232	Salida de datos del puerto RS-232 hacia el PC.
Pin 8, 9 y 10	Señales que activa los leds de control D5, D6 y D7.

EL PIC 16F727 se presenta en un integrado tipo DIP de 40 pins y está construido con la tecnología *nanoWatt XLP* de ahorro de energía de *Microchip*. Para ver sus principales

características se recomienda leer las hojas de especificaciones mediante el link a la página web del fabricante disponible en el Anexo F, pero unas de las más importantes son: 14KB de memoria flash para el programa, 5MIPS de velocidad de CPU, oscilador interno de hasta 16MHz, una entrada USART (para RS-232) y una SPP (para I²C o SPI), tres *timers* (dos de 8-bits y uno de 16-bits), conversor ADC de 14 canales y 8-bits, 25mA suministrable por los pins de salida, *Watchdog Timer* y *In-Circuit Serial Programming* (ICSP) para poder programarlo con el accesorio MPLAB ICD 2.

Más adelante en el apartado *Firmware* se explicará con más detalle el software diseñado y cómo se programa el microcontrolador desde el PC sin necesidad de extraerlo del hardware.

Para terminar este apartado, destacar que se ha asignado un nombre a cada nodo de tensión ya que el programa *Altium Designer 6* une los nodos con nombres coincidentes sin necesidad de que exista un dibujo de conexión física (*wire*) entre ellos. Esto facilita el diseño esquemático y hace que la presentación del mismo sea más nítida. Esta técnica también es muy útil para conectar nodos de diferentes módulos sin necesidad de poner entradas y salidas, como por ejemplo en el caso de Vcc o de GND.

4.2.7. Etapa lógica

Antes de atacar los transistores de potencia con las salidas del microcontrolador, se ha insertado una etapa lógica que evita la activación de fases opuestas en el motor. Bien es cierto que el firmware del microcontrolador controla en todo momento la activación de las fases del motor, pero por si hubiera un fallo en el microcontrolador o para los estados de transición (reset o inicio de la alimentación) se ha diseñado esta etapa.

El diseño se basa en que el motor tiene cuatro fases: A, B, C y D. Las fases están emparejadas dos a dos de la siguiente forma: A y C, B y D. Si la fase A está activa la fase C no puede estarlo y viceversa. Y lo mismo ocurre con las fases B y D.

Para controlar 4 fases sólo harían falta 2 pins con un bit asociado a cada pin ($2^2 = 4$), pero para asegurar que no ocurren transiciones extrañas se han introducido dos bits más. Estos bits hacen de activación o *enable* de las fases. Además, de esta forma podemos dejar todas las salidas inactivas para ahorrar energía y sincronizar la activación de las fases con mejor fiabilidad.

Así encontramos un pin que controla las fases A y C llamado AC y su respectivo *enable* llamado ENAC. Y simétricamente encontramos un pin BD que controla las fases B y D con su respectivo *enable* ENBD.

Después de la etapa lógica ya encontramos las cuatro señales que actuarán sobre los transistores de potencia previo paso por los optoacopladores para aislar el circuito de potencia de la etapa lógica.

Tabla 5: Ecuaciones lógicas de las fases del motor.

Ecuaciones lógicas
$A = ENAC \cdot AC$
$B = ENBD \cdot BD$
$C = ENAC \cdot \overline{AC}$
$D = ENBD \cdot \overline{BD}$

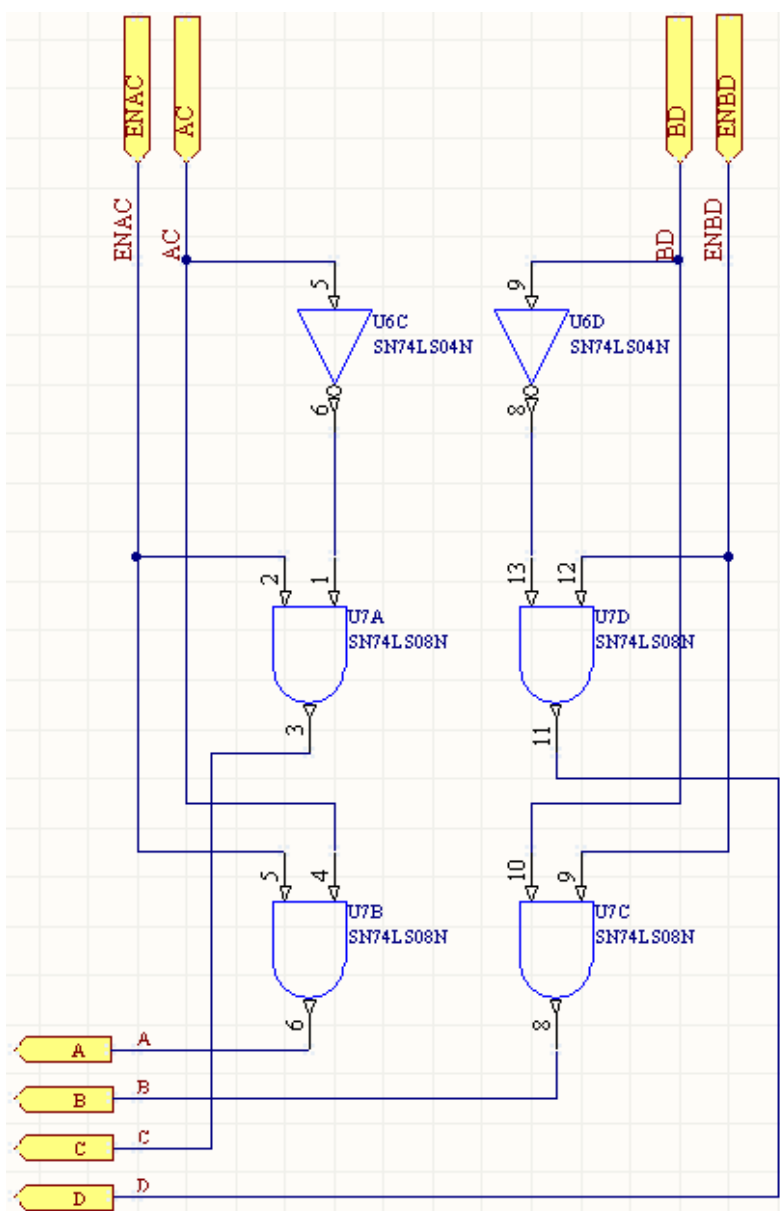


Fig. 45: Esquema eléctrico de la etapa lógica.

Tabla 6: Tabla de verdad de las fases del motor.

Fase Activa	ENAC	ENBD	AC	BD
A	1	X	1	X
C	1	X	0	X
B	X	1	X	1
D	X	1	X	0

Como se ve en la *Tabla 6*, para que una fase esté activa es necesario que el *enable* de esa fase esté activo. Entonces dependiendo del valor que tome el campo de control de las fases, se activará una u otra dependiendo de su valor (1 o 0).

Por ejemplo, si se desea activar la fase A, primero se pondrá el bit de control de fase AC a 1, y posteriormente se activará el ENAC. Si posteriormente se desea activar la fase C, primero se desactiva el ENAC, luego se pone AC a 0 y se vuelve a activar ENAC. De esta forma no hay ninguna posibilidad de que las fases A y C estén activas a la vez.

El motor escogido permite realizar un paso entero o medio paso. Para realizar medio paso es necesaria la activación simultánea de dos fases contiguas, como por ejemplo la A y la B. Entonces el rotor gira sólo medio paso, en este caso $3,75^\circ$. La sincronización en la activación de las fases es imprescindible al realizar medio paso y es aquí cuando los *enables* toman mayor relevancia. Así, una vez configurados los pins de las fases (AC y BD) se activan los pins de *enable* ENAC y ENBD sincronamente asegurando un correcto giro del rotor.

Como puede verse en la *Fig. 45* se ha utilizado lógica positiva ya que se disponía de integrados AND y NOT, pero también se hubiera podido utilizar lógica con puertas NAND o NOR tras una simple conversión.

En la *Fig. 46* se puede ver la equivalencia de las puertas estándares (NOT, AND y OR) a lógica NAND. La ventaja de utilizar este tipo de lógica es que solamente se utilizan puertas del mismo tipo, en este caso NAND. El inconveniente es que suele incrementar el número de puertas a utilizar.

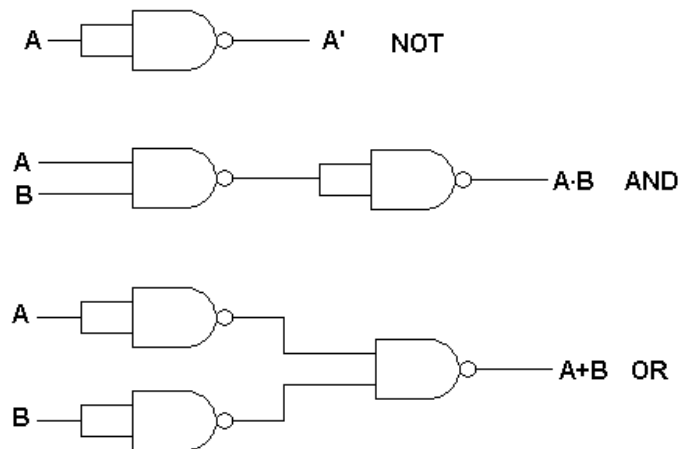


Fig. 46: Equivalencia en lógica NAND de las puertas estándares.

4.2.8. Optoacopladores y transistores de potencia

La última etapa antes de atacar las fases del motor es la de aislamiento y potencia. En este módulo se aísla la etapa de potencia y la lógica mediante optoacopladores y posteriormente se entrega potencia a las fases del motor con transistores capaces de entregar hasta 1A de corriente y los 12V necesarios para que el motor gire con suficiente par.

El optoacoplador elegido ha sido el 4N27 ya que se pudieron conseguir muestras gratuitas en la página de *Texas Instruments*. Es un optoacoplador muy utilizado durante años en la industria aunque ya un poco en desuso debido a su larga vida y la mejora de prestaciones de los nuevos modelos. De todas formas cumple de sobras con los requisitos técnicos del proyecto.

Para polarizar la corriente a través del led emisor y del fototransistor, se han elegido resistencias que mantienen la corriente en los 10mA tanto en la parte aislada como en la de ataque a los transistores ya que en este rango la transferencia de corriente (CTR) es máxima.

Como transistor de potencia se ha elegido el UC3710T capaz de entregar hasta 1A de corriente y trabajar en un rango de tensiones de 5V a 18V DC. Al igual que los optoacopladores, se han podido conseguir muestras gratuitas en la página web de *Texas Instruments*.

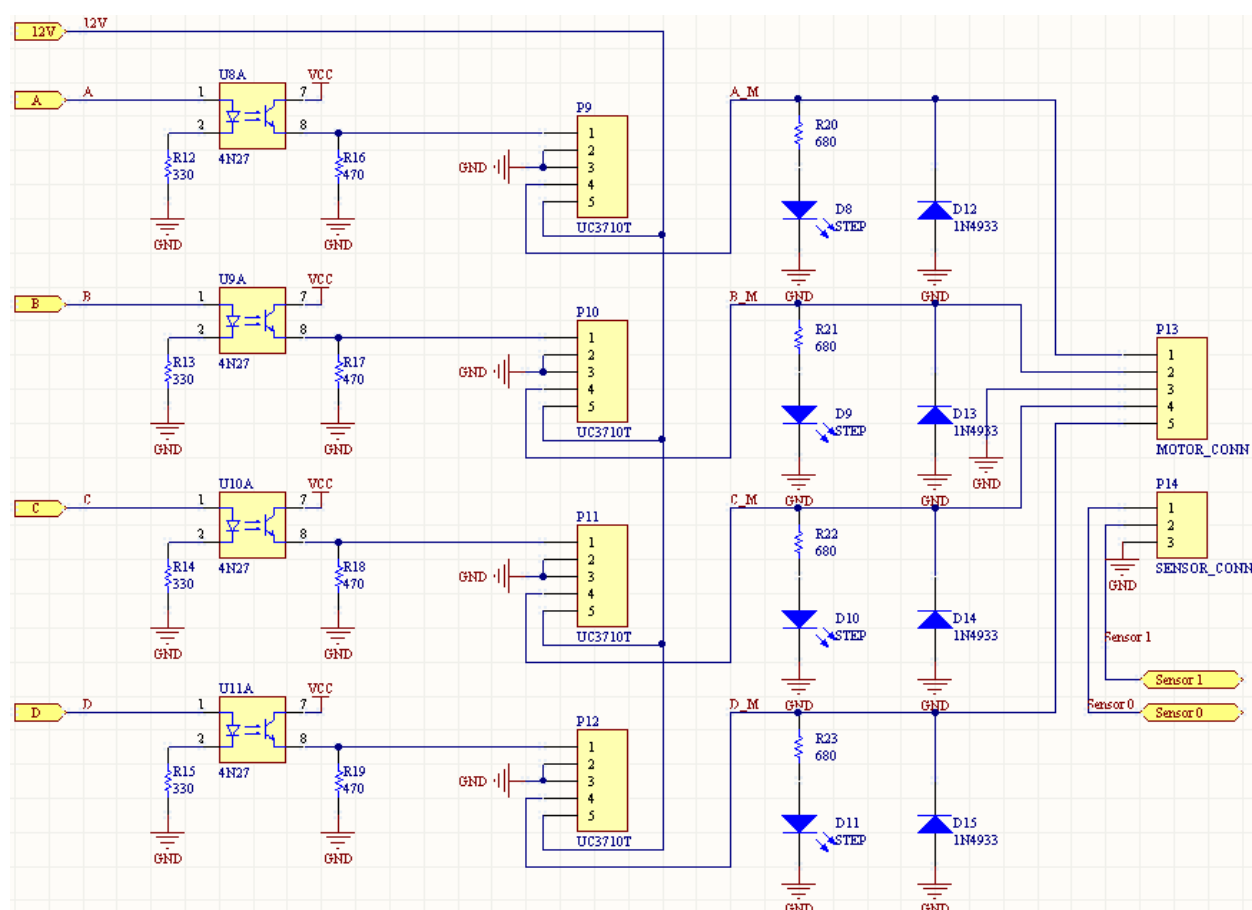


Fig. 47: Esquema eléctrico del aislamiento con optoacopladores, transistores de potencia, diodos de recuperación rápida y conectores del motor y los interruptores.

Los transistores trabajan de la siguiente forma: cuando se detecta nivel alto en la parte aislada se cierra el transistor de la parte de potencia entregando 12V DC y la corriente necesaria. Cada una de estas salidas ataca directamente el bobinado de la fase correspondiente. También se han

introducido unos leds de visualización (D8, D9, D10 y D11) para observar que realmente se alimenta la fase y poder hacer pruebas sin necesidad de conectar físicamente el motor.

Y para evitar que los bobinados de las fases del motor se queden energizados una vez se ha quitado la alimentación, se ha insertado un diodo de recuperación rápida (1N4933) para darle salida a esta energía a través del GND.

Finalmente se encuentran las conexiones de las fases del motor y el GND en el conector P13 y las conexiones de los interruptores del hardware en el conector P14.

En un principio, en cada uno de los nodos Sensor0 y Sensor1 se tenía que conectar un final de carrera para determinar cuando el motor estuviese en la posición más baja y en la más alta y así parar de girar el tornillo de sintonización. Más adelante y tras construir el soporte del motor, se vio que el movimiento era mínimo (1cm aprox.) y la fijación de los finales de carrera no era trivial. Por eso se cambió su funcionalidad y se hizo un pulsador de paro general y un pulsador que hace girar el motor un paso a la izquierda. Entonces el control de la altura del motor se realiza desde el software relacionando la frecuencia mínima y máxima con la penetración del tornillo dentro de la cavidad resonante.

4.2.9. Elaboración física del hardware

Una vez realizados todos los esquemáticos, primeramente se unen en otro esquemático que contiene cada uno de los bloques anteriores y se interconectan las entradas y salidas de cada módulo. Posteriormente se crea un documento PCB en donde ya aparecen los *footprints* de los dispositivos para disponerlos en la superficie necesaria y hacer el *routing* y los *layouts* para la construcción del hardware.

Una vez ubicados correctamente los componentes sobre la placa y realizado el *routing*, se generan los *layout* de las caras superior e inferior y se imprimen en un papel transparente. En este papel se imprime en color negro los sitios de la placa que tienen pistas o cobre. Seguidamente se alinean los *layouts* superior e inferior y en medio de los dos se inserta la lámina de cobre. Se alinean los dos *layouts* mediante las cruces de alineación y se aplica un proceso fotolítico que quita la protección del cobre de las zonas no impresas. Posteriormente se sumerge la lámina en una solución química que corroe el cobre expuesto al proceso fotolítico y deja intacto el resto.

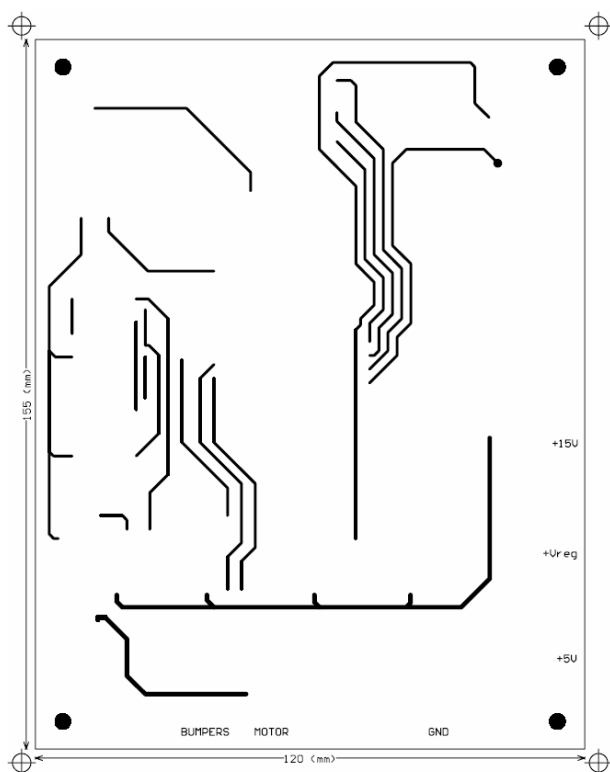


Fig. 48: Layout de la cara superior del PCB.

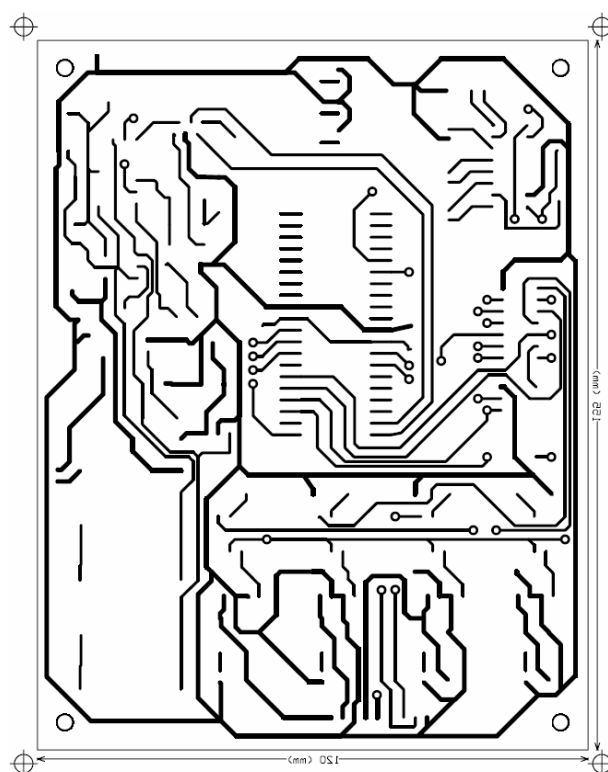


Fig. 49: Layout de la cara inferior del PCB con efecto espejo.

En la Fig. 48 se puede ver el layout de la cara superior del PCB. Se ha intentado que hubiera el mínimo de pistas en la cara superior ya que en ésta es en donde se ubican todos los componentes y así evitar posibles cortocircuitos.

En la Fig. 49 se ve el layout de la cara inferior del PCB pero impreso con efecto espejo. De esta forma, si se superponen los dos *layouts* se hace coincidir a la perfección los agujeros de los componentes y en medio se inserta la lámina de cobre para ser tratada.

Tras estas operaciones se obtiene el PCB diseñado pero sin agujeros. Para hacer los agujeros hay dos formas: manual o automática. La manual se basa en coger el PCB, un taladro y mucha paciencia. La automática consiste en generar un archivo de agujeros e insertarlo en un taladro con control CNC (*Computed Numerically Controlled*). En este proyecto se tenían que hacer más de 300 agujeros y se decidió hacerlo de forma automática, evidentemente.

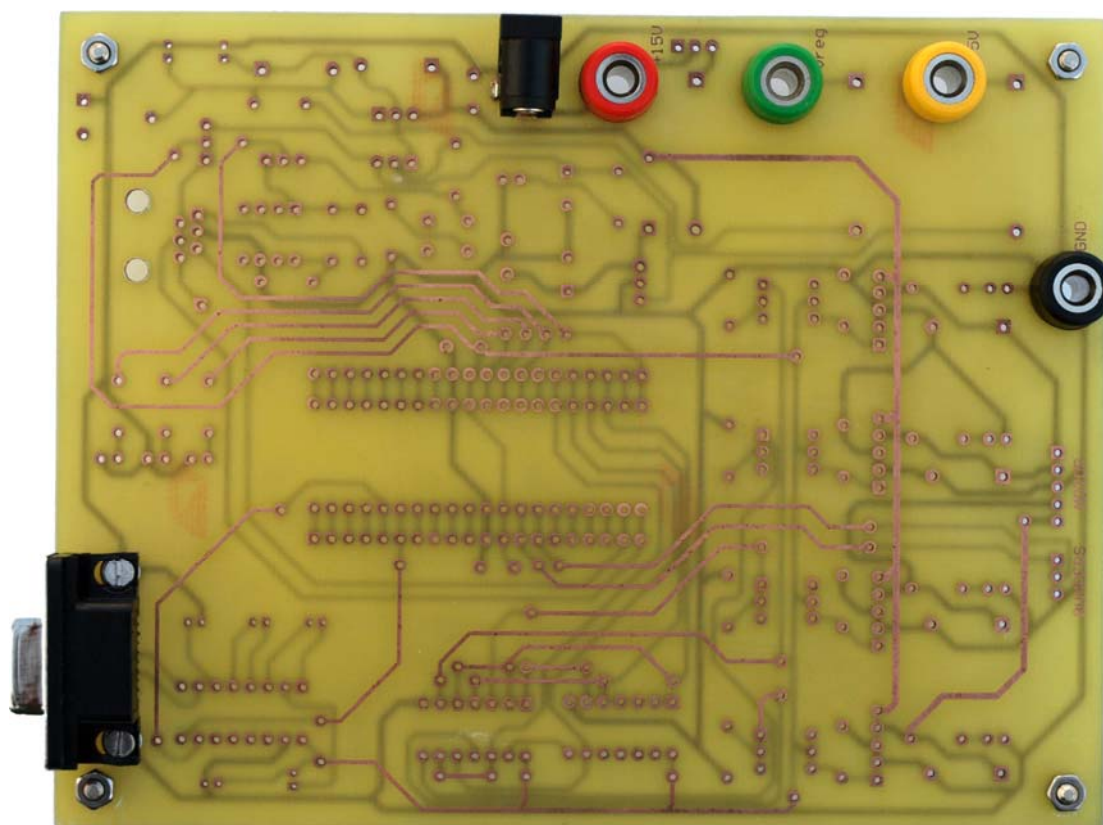


Fig. 50: Vista de la cara superior del PCB.

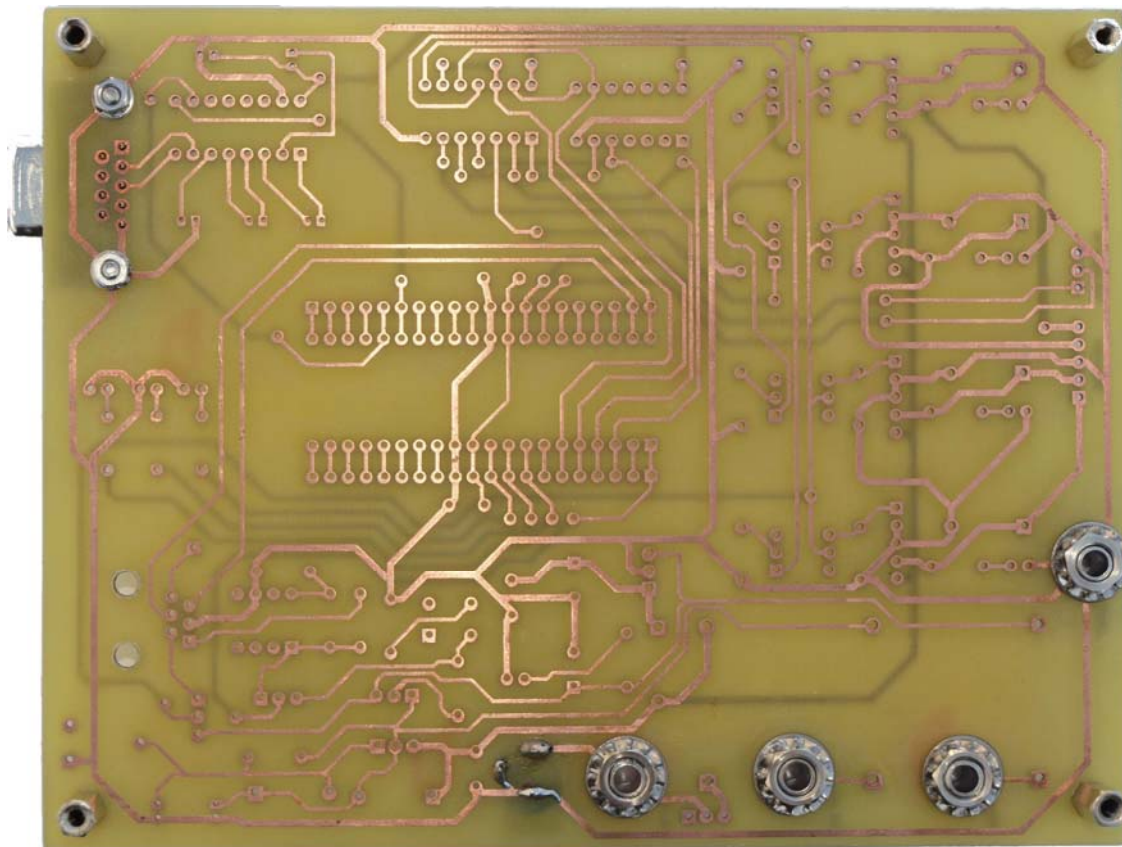


Fig. 51: Vista de la cara inferior del PCB.

Para finalizar, solamente hace falta soldar todos los componentes y cerciorarse del correcto funcionamiento de todo el hardware testeando diferentes puntos clave.

En la *Fig. 52* de la página siguiente, se puede ver una fotografía del hardware ya finalizado y todos los componentes soldados. En la parte central está el microcontrolador con los pins de expansión a cada lado. A su izquierda se encuentran los integrados lógicos (NOT y AND). Arriba se encuentra el bloque RS-232, con el conector DB-9, el integrado MAX232 y sus condensadores. A su derecha los tres leds verdes de información y más a la derecha el conector RJ-12 para la programación en circuito del microcontrolador (ICSP) y el integrado TL431 junto con el potenciómetro de regulación de tensión de V_{reg} y los distintos *jumpers* e interruptores.

A la derecha del microcontrolador tenemos el integrado LM317 (con disipador) que se encarga de fijar la tensión V_{reg} . Debajo el jack de 12V DC y a su izquierda los dos condensadores y el regulador de tensión a 5V L7805. Más abajo encontramos los conectores banana con sus respectivos fusibles de protección y a su izquierda la etapa de potencia con los cuatro transistores UC3710 con grandes disipadores para refrigerarlos. Entre las patas de los disipadores se encuentran los optoacopladores 4N27 de cada fase que aíslan la etapa de potencia y la de control.

Debajo de los transistores se encuentran los leds rojos indicadores de cada fase y los diodos de recuperación rápida 1N4933 antes de dar salida a las señales por el conector de cinco pins hacia el motor. Y a su izquierda se encuentran los dos pulsadores: el de paro y el de un paso a la izquierda.

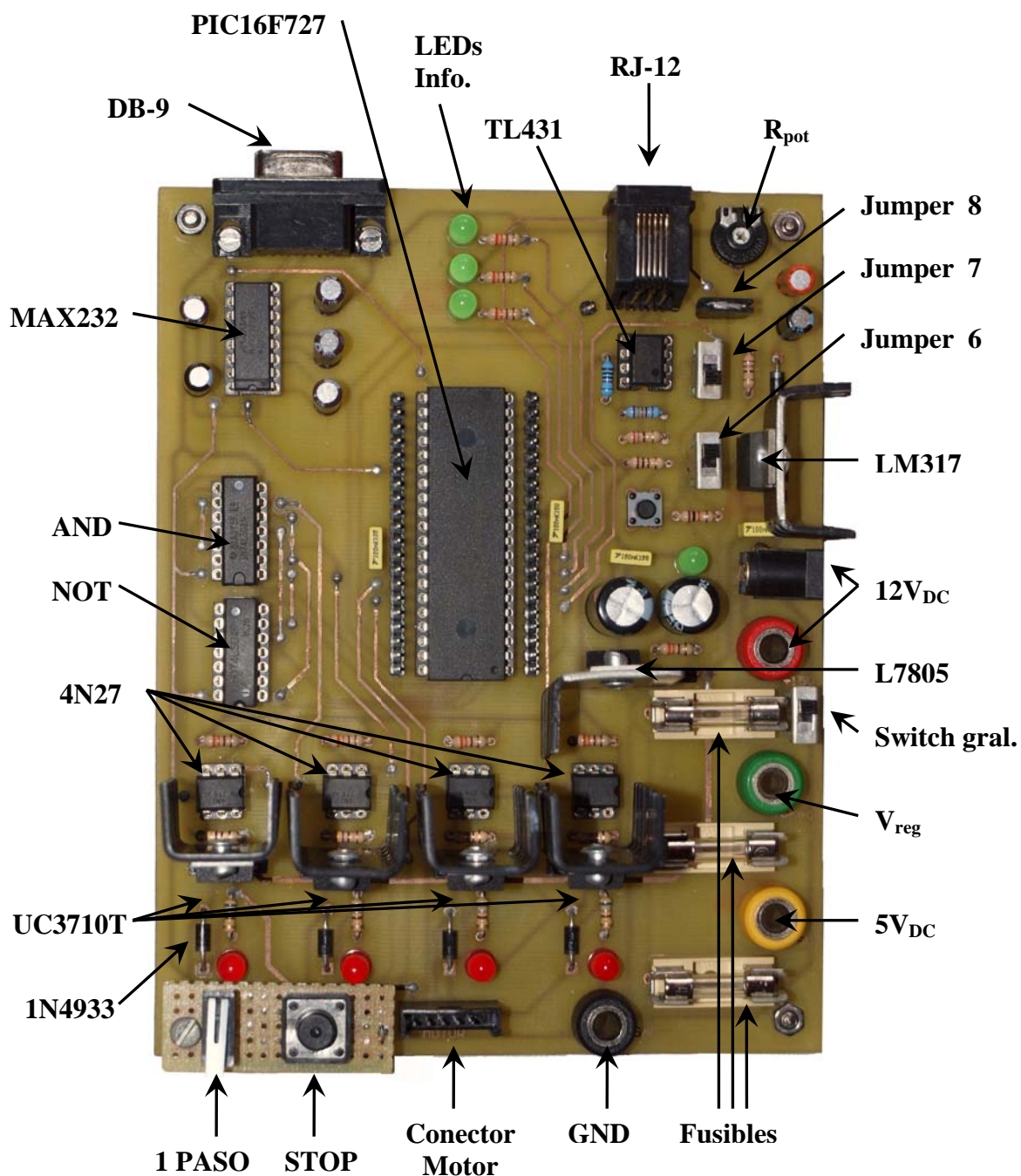


Fig. 52: Imagen del hardware finalizado.

4.3. Software

El software de control se ha implementado en entorno Matlab ya que se disponía de las funciones que leen los datos del analizador de redes y están desarrolladas en este mismo entorno. También se ha tenido en cuenta la gran capacidad de cálculo y facilidad de presentación de resultados mediante gráficos que ofrece Matlab así como el manejo de variables complejas.

Se ha diseñado un archivo que lleva el hilo principal del control y realiza la gestión de errores. Es este mismo archivo el encargado de llamar a las diferentes funciones en que se ha dividido el software y actualizar las variables globales.

A continuación se comentará el funcionamiento de los diferentes archivos y funciones relacionándolos con el hardware diseñado.

4.3.1. Descripción funcional del software

El software diseñado se encarga de controlar todo el auto ajuste del resonador mandando las órdenes precisas al microcontrolador y leyendo la información de la medida del analizador de redes.

El programa se inicia tras llamar la función *main_sintonizacion* desde la línea de comandos del Matlab y teniendo como directorio de trabajo el directorio en donde se encuentran todos los archivos de la aplicación.

Los *reports* de la aplicación van apareciendo en la línea de comandos y las gráficas aparecen en nuevas ventanas. Los datos que se piden al usuario también deben ser introducidos en la línea de comandos mediante el teclado.

Cuando se ejecuta la función *main_sintonizacion*, primero se guía al usuario en la ubicación del resonador de microondas en el soporte y la correcta conexión del motor y el hardware. Posteriormente se piden al usuario datos necesarios como el rango de frecuencias del analizador de redes y el número de puntos de la medida, su dirección GPIB, frecuencia a la que centrar el resonador, los directorios de trabajo, etc.

Una vez introducidos los datos, se inicia el autoajuste del resonador. Primeramente se mide la frecuencia a la que está centrado para decidir si se debe subir o bajar de frecuencia. Una vez sabido el sentido de giro del motor, se inicia la sintonización realizando pasos enteros hasta que se sobrepasa la frecuencia deseada. Entonces se cambia el sentido de giro del motor y se realiza medio paso. En este punto se analiza cual de las frecuencias se acerca más a la frecuencia deseada por el usuario y el software se encarga de sintonizarla.

Una vez sintonizado a la frecuencia más próxima a la deseada por el usuario, se muestra por pantalla la frecuencia sintonizada y aparecen las gráficas de la sintonización y los parámetros S_{21} y S_{11} .

Para finalizar se guardan las variables y se termina la comunicación con el microcontrolador liberando el puerto serie del PC configurado en el Matlab.

Esta es a grandes rasgos la columna vertebral del software pero a continuación se describirá uno a uno cada función del software de control.

4.3.2. Programa principal: *main_sintonizacion*

El archivo *main_sintonizacion.m* es el hilo principal del programa de control y se encarga de llamar a las demás funciones para lograr la sintonización automática, la corrección de errores y abrir y cerrar la comunicación con el hardware.

Primeramente se declaran todas las variables globales ya que las diferentes funciones las irán modificando de manera no concurrente. Se ha optado por esta técnica ya que en ningún momento hay dos funciones accediendo a la misma variable y así no hace falta pasar las variables como parámetro en las llamadas de las funciones.

Seguidamente hay un texto descriptivo que explica como conectar la alimentación del hardware y como emplazar el filtro de microondas en su posición correcta. En una primera fase se configura el puerto serie y se inicia la comunicación con el microcontrolador mediante la función *inicializar_puerto*. Si la conexión ha sido correcta se piden al usuario los parámetros del analizador de redes y la frecuencia a la que se desea sintonizar el filtro mediante la función *peticion_datos*. Una vez introducidos los datos, se pide al usuario un modo de sintonización. Existen tres modos de operabilidad del software:

- *Sintonización automática con parametrización previa del resonador*: en esta opción primero se calculan las frecuencias que puede sintonizar el resonador y se guardan en un archivo cuyo nombre es introducido por el usuario. Posteriormente se procede a la sintonización automática a la frecuencia deseada, siempre y cuando se encuentre dentro del rango sintonizable por el resonador y medible por el analizador de redes.
- *Sintonización automática con carga de los parámetros del resonador*: en esta opción se cargan las frecuencias sintonizables por el resonador de un archivo guardado previamente (con el modo anterior de parametrización) y posteriormente se procede a la sintonización automática siempre y cuando se esté dentro del rango.
- *Sintonización automática sin parámetros*: en esta opción, el ingeniero debe asegurarse que el resonador es capaz de sintonizar la frecuencia deseada y que ésta se encuentra dentro del rango de frecuencias medibles por el analizador de redes. Es la opción más

práctica y rápida siempre y cuando se tenga un conocimiento de las frecuencias sintonizables por el resonador.

Una vez elegido el modo de operabilidad, se carga, si es necesario, el archivo que contiene los parámetros del resonador o se llama la función *inicializar_filtro* que se encarga de parametrizarlo. Más adelante se comentarán las diferentes funcionalidades de este código.

Una vez sabida la frecuencia máxima y mínima sintonizables por el resonador y el rango de frecuencias de medida del analizador de redes, se procede al primer cálculo de la frecuencia de resonancia. Esta frecuencia es a la que está sintonizado actualmente el resonador y sirve para saber si se tiene que subir o bajar de frecuencia para llegar a la frecuencia deseada.

Posteriormente se empieza la sintonización con pasos enteros para ir más rápido hacia la frecuencia deseada y ya al final se pasa a medio paso para conseguir mayor precisión en la sintonización. Se ejecutan pasos enteros hasta que la frecuencia actual del resonador supera la frecuencia deseada (f_c) y entonces se hace medio paso en sentido contrario (ver Fig. 53). En este punto la frecuencia deseada se encuentra entre estas tres frecuencias: la superior (f_{n+1}), la inferior (f_{n+2}) y la intermedia (f_{n+3}). El programa calcula la diferencia entre cada una de estas tres frecuencias y la deseada y escoge la que se encuentra más próxima.

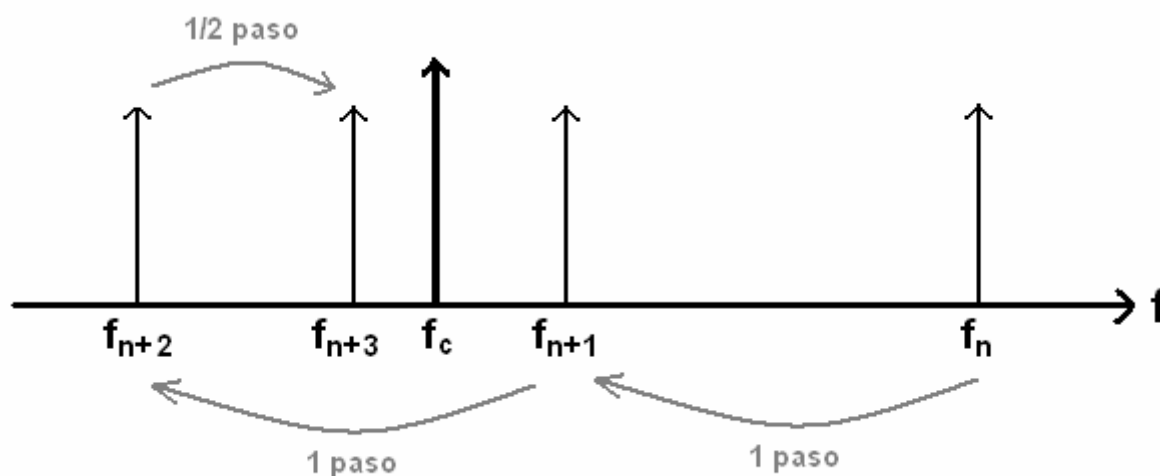


Fig. 53: Representación del proceso de sintonización de una frecuencia f_c con $n+3$ pasos.

Como se puede ver en la Fig. 53 el software de sintonización realiza pasos enteros hacia la frecuencia deseada de sintonización (f_c) hasta que en el paso $n+2$ se sobrepasa. Entonces realiza medio paso en sentido contrario y evalúa cual es la menor diferencia entre las frecuencias de los pasos $n+1$, $n+2$ y $n+3$ y la frecuencia a sintonizar f_c . Una vez evaluadas las diferencias escoge la frecuencia más próxima a la frecuencia deseada de sintonización haciendo medio paso hacia arriba, o hacia abajo, o quedándose en el mismo sitio para la frecuencia intermedia.

Destacar que en todo momento se controla que la frecuencia de resonancia no exceda el rango de frecuencias sintonizables por el resonador ni el rango de medida del analizador de redes.

Una vez sintonizado el filtro se muestra por pantalla la frecuencia sintonizada y unos gráficos con el módulo del parámetro S_{11} y S_{21} tal y como se muestra en la Fig. 54.

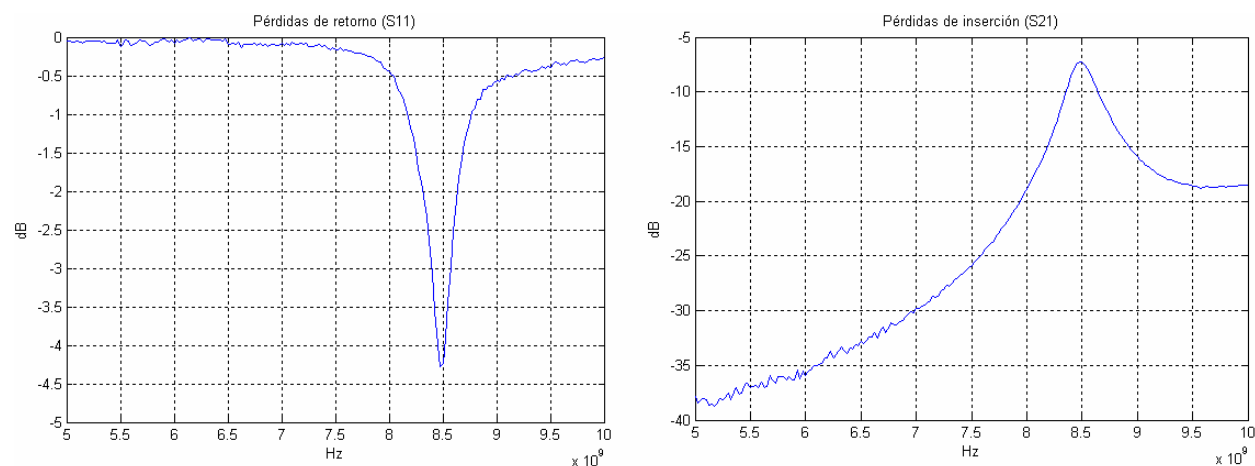


Fig. 54: Gráficas del módulo de los parámetros S_{11} y S_{21} a 8,5GHz del resonador sintonizable utilizado.

También aparece una imagen con los gráficos de: frecuencia de resonancia en cada paso, el incremento de frecuencia de resonancia entre pasos y las pérdidas de inserción en cada paso realizado durante la sintonización (ver Fig. 55). Con estas gráficas se observa de forma clara la evolución de la frecuencia de resonancia en cada paso.

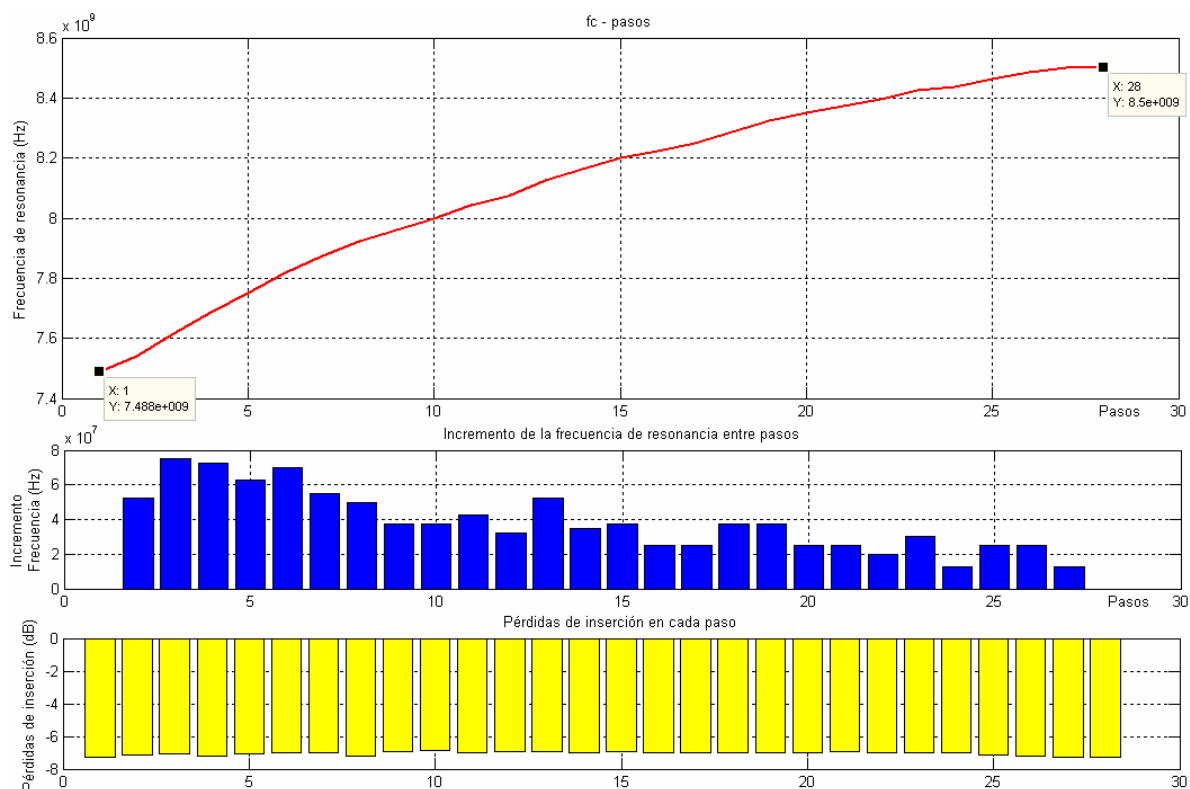


Fig. 55: Frecuencia de resonancia en cada paso, incremento de frecuencia de resonancia entre pasos consecutivos y pérdidas de inserción en cada paso de una sintonización del resonador utilizado.

Para terminar se cierra la comunicación serie con el hardware y se guardan los parámetros del resonador en el archivo creado. También se almacenan todos los parámetros de la sintonización en un archivo llamado *ultima_sintonizacion.mat* por si fuera necesario extraer otras gráficas.

Al final del archivo se encuentra toda la gestión de errores. Para controlar todos los errores posibles se ha modificado el *handle* de errores y se han gestionado uno a uno. Si en algún momento surge un error que no esté controlado en el *handle*, éste se muestra por pantalla para que el usuario reciba toda la información posible y pueda solucionarlo.

Véase el código de la función en el *Anexo B*. Se recomienda que primero se haga una lectura no muy profunda haciendo hincapié en las líneas y palabras resaltadas en negrita ya que conforman el cuerpo principal del archivo y de esta forma se comprenderá mejor la estructura y funcionalidad del mismo. Para mayor ayuda se puede releer este apartado siguiendo la explicación del archivo e identificando cada apartado con su código correspondiente.

4.3.3. Configuración del puerto serie: *inicializar_puerto*

Esta función se encarga de crear un objeto tipo puerto serie y configurarlo para la correcta comunicación con el microcontrolador (ver *Tabla 7*). Posteriormente establece la comunicación con el microcontrolador y si éste responde adecuadamente se prosigue con el programa principal. Si no responde se suspende la ejecución del hilo principal y se lanza un mensaje de error.

Tabla 7: Configuración del puerto serie.

Baud rate	Data bits
9600	8
Parity	Stop bits
No	1
Flow Control	
No (software)	

Para abrir la comunicación se ha establecido el siguiente protocolo. Primeramente el PC envía al microcontrolador el mensaje 1000 0000_b en binario, o 128_d en decimal. Para que el microcontrolador responda correctamente tiene que estar en modo ‘espera de iniciar comunicación’ o disponer ya de una comunicación con el PC y esperar el envío de comandos. En estos dos casos el microcontrolador responde con el mismo mensaje. Véase *Tabla 8* y *Tabla 9*.

Tabla 8: Codificación de la instrucción de iniciar la comunicación con el microcontrolador.

Instrucción de PC a microcontrolador	Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)							
			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
Iniciar comunicación	128	0x80	1	0	0	0	0	0	0	0

Si el microcontrolador responde algún mensaje que no sea el anterior, se cancela la comunicación y se termina la ejecución del hilo principal lanzando un mensaje de error. Si por el contrario el microcontrolador no responde, salta un mensaje de error debido al *timeout* de la instrucción de lectura del puerto serie y se termina la ejecución del hilo principal. Véase *Tabla 9*.

Tabla 9: Codificación de la instrucción de comunicación iniciada correctamente por el microcontrolador.

Instrucción de microcontrolador a PC	Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)							
			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
Comunicación iniciada OK	128	0x80	1	0	0	0	0	0	0	0

Véase el código de la función en el *Anexo B*.

4.3.4. Petición de datos al usuario: *peticion_datos*

Esta función pide los datos al usuario referente al analizador de redes (frecuencia mínima y máxima de medida, dirección GPIB y número de puntos de la medida), la frecuencia a la que se desea sintonizar el resonador y el directorio y el nombre de archivo en donde guardar los parámetros del resonador.

Para tener todos los datos fácilmente disponibles se crea un directorio llamado *sintonización_filtros* dentro del directorio de trabajo actual. En dicho directorio se almacenará toda la información y variables referentes al resonador sintonizado dentro de un archivo que lleva el nombre del filtro que se desee (por ejemplo *.\sintonizacion_filtros\resonador.mat*). En este directorio también se almacenaran todas las variables de la última sintonización para una posible revisión posterior en un archivo llamado *ultima_sintonizacion.mat* (ubicación: *.\sintonizacion_filtros\ultima_sintonizacion.mat*).

Destacar que se ha controlado en todo momento que los directorios en donde se guardan los parámetros del filtro existan, y en caso de no existir se crean.

Véase el código de la función en el *Anexo B*.

4.3.5. Parametrización del filtro: *inicializar_filtro*

Esta función se encarga de extraer las frecuencias mínima y máxima sintonizables por el resonador para saber el rango en que se puede sintonizar. En el diseño inicial se decidió poner dos finales de carrera, uno superior y otro inferior, de tal manera que cuando se activara uno de ellos el motor no pudiera dar más pasos en ese sentido y evitar dañar el dispositivo. A medida que se iba implementando el soporte, se descartó esta opción ya que la precisión de los finales de carrera convencionales era muy pequeña para el movimiento que realiza el motor.

Entonces se buscó la manera de calcular la frecuencia máxima y mínima que puede sintonizar el resonador y se diseñó esta función. Su funcionamiento es el siguiente: debe situarse el tornillo de sintonización en su posición más baja (lo más introducido posible dentro del orificio de sintonización) pero cerca del rango en que empieza a variar la frecuencia. Si no se sitúa cerca del rango de variación, se puede dar el caso que al no variar la frecuencia en un cierto número de pasos el software piense que ha llegado a la frecuencia superior y detenga el proceso sin haber parametrizado el resonador.

A continuación la función va ejecutando pasos enteros subiendo de frecuencia hasta detectar que la frecuencia no varía durante un cierto número de pasos. Entonces se decide que esa frecuencia es la máxima sintonizable por el filtro ya que no aumenta más y termina el proceso de parametrización.

El número de veces que debe repetirse la misma frecuencia para decidir que ya no es posible sintonizar frecuencias superiores viene establecido por la variable *repeticion*, que se ha establecido en 6, pero puede cambiarse su valor en la línea 23 del archivo *inicializar_filtro.m*.

Para finalizar esta función se muestran las gráficas de parametrización y se guardan las variables en el archivo del filtro.

Véase el código de la función en el *Anexo B*.

4.3.6. Cálculo del parámetro S_{21} : *calcular_S*

Esta función se encarga de extraer el parámetro S_{21} del resonador. Se basa en las dos funciones previas *enviar* y *rebre* que se comunican con el analizador de redes mediante bus GPIB y obtienen los puntos de la medida en formato complejo. Posteriormente se genera una matriz que tiene como primera columna las frecuencias de medida del analizador de redes y como segunda columna el módulo del parámetro S_{21} expresado en dB. Ya posteriormente se mide el módulo de las pérdidas de retorno (S_{11}) y se añade a la matriz en la tercera columna.

Originalmente se medían los cuatro parámetros S del resonador y se conformaba toda la matriz S de módulos expresados en dB (de ahí el nombre de la función). Pero para ahorrar tiempo de medida se decidió calcular únicamente el parámetro deseado, el S_{21} , y ya posteriormente calcular

el S_{11} para representarlo en las gráficas. De esta forma se disminuye en $\frac{3}{4}$ el tiempo de medida llegando a su valor mínimo de 1 segundo aproximadamente.

Véase a en el *Anexo B* el código de la función *calcular_S.m* y las funciones de obtención de datos del analizador de redes *enviar.cpp* y *rebre.cpp*. Estas dos últimas funciones no serán comentadas ya que no se han diseñado en este proyecto y para entender su funcionalidad se tendría que explicar con detalle el funcionamiento del protocolo de comunicación del bus GPIB.

4.3.7. Cálculo de la frecuencia central del resonador: *calcular_fc*

Esta función calcula la frecuencia central o de resonancia del resonador como la frecuencia intermedia de la banda de paso a -3dB. Primeramente se calcula el punto de máxima transferencia de potencia del parámetro S_{21} (o el de mínima atenuación). Seguidamente se obtienen las frecuencias de corte superior e inferior a -3dB del máximo y se calcula la frecuencia de resonancia como la frecuencia intermedia entre las frecuencias de corte. También se mide el ancho de banda de la banda de paso a -3dB (ver 2.2.4 *Ancho de banda* y 2.2.5 *Frecuencia central*)

Véase el código de la función en el *Anexo B*.

4.3.8. Cálculo de la frecuencia central del resonador en modo simulación: *calcular_fc_simulacion*

Para comprobar el correcto funcionamiento del software y del hardware es necesario disponer de un analizador de redes y de un filtro de microondas o resonador sintonizable. En muchas ocasiones no era posible acceder al analizador de redes ya que se trata de un instrumento de medida muy caro y habitualmente estaba ocupado por profesores y doctorantes. Así que se decidió crear una función auxiliar que simula la obtención de la frecuencia central de un supuesto filtro de microondas o resonador.

Para empezar, se crea un vector de frecuencias comprendido entre el rango de medida del analizador de redes con incrementos de frecuencia constantes (el vector *fcv*). De esta forma se consigue crear un vector de frecuencias que simula la frecuencia a la que se centra el filtro en cada paso. Así, si se realiza un paso a la izquierda, se incrementa dos posiciones el puntero del vector y se obtiene una frecuencia dos posiciones superior. Si sólo realizamos medio paso, únicamente se incrementa el puntero en una posición.

Este vector consta de 300 posiciones equidistantes entre la frecuencia mínima y máxima de medida del analizador de redes, pero este valor puede modificarse directamente sobre el código en la línea 111 del archivo *main_sintonizacion.m* dependiendo de la precisión que se desee.

El puntero del vector se ha denominado *puntero_fcv*, y está inicializado en el punto medio del vector por razones que se explican en el siguiente párrafo. Si se desea modificar este valor, puede hacerse en la línea 115 del archivo *main_sintonizacion*.

Para poder simular sintonizaciones de frecuencias superiores e inferiores respecto a la frecuencia actual del supuesto filtro, la frecuencia a la que se inicia la sintonización es la que se encuentra en la mitad del vector *fcv*. De esta forma, si se escoge una frecuencia a sintonizar inferior a la intermedia del vector, el motor gira en sentido diestro. Y si se elige una frecuencia superior, el motor gira en sentido zurdo.

Cada vez que se llama a la función *calcular_fc_simulacion*, se monitorizan los valores de las variables *Modo* y *Direccion*, y en función de su valor se incrementa o decrementa el puntero en una o dos posiciones y se devuelve la frecuencia correspondiente a esa posición. Véase la *Tabla 10*.

Tabla 10: Incremento del puntero del vector de frecuencias *fcv* en modo simulación.

Modo	Direccion	<i>puntero_fcv</i>
Half	Izquierda	<i>puntero_fcv</i> +1
Half	Derecha	<i>puntero_fcv</i> -1
Full	Izquierda	<i>puntero_fcv</i> +2
Full	Derecha	<i>puntero_fcv</i> -2

Para operar en modo simulación, el único cambio que debe realizarse en el código es sustituir las dos líneas de código que se encargan de calcular el parámetro S_{21} (*calcular_S*) y hallar la frecuencia central del filtro (*calcular_fc*) por la llamada a la función *calcular_fc_simulacion*.

Cabe destacar que en modo simulación no es posible realizar la parametrización del filtro, ya que en ningún momento se repiten frecuencias y llegaríamos a la frecuencia superior que coincide con la máxima frecuencia medible por el analizador de redes.

Gracias a esta función se han podido simular todos los cambios efectuados en el software sin necesidad de disponer de un analizador de redes ni un filtro de microondas sintonizable. Y como el hardware dispone de leds que permiten ver la activación de las fases del motor sin necesidad de conectarlo, se han podido llevar a cabo exhaustivas pruebas en las diferentes revisiones del software.

Véase el código de la función en el *Anexo B*.

4.3.9. Enviar órdenes al microcontrolador: *envia_orden*

Esta función se encarga de enviar las órdenes de giro al microcontrolador y recibir la confirmación si éstas se han ejecutado correctamente o de lanzar un error si las órdenes no han sido ejecutadas correctamente.

El programa principal dispone de dos variables *Modo* y *Direccion* que establecen el tipo de giro del motor. Cuando se llama la función *envia_orden*, ésta mira el valor de las dos variables anteriores y envía al microcontrolador el código de giro correspondiente (ver *Tabla 12*). Posteriormente espera la confirmación del microcontrolador diciéndole si el giro se ha realizado correctamente o si por el contrario no se ha podido girar el motor (ver *Tabla 13*). Si por algún motivo el código que recibe el microcontrolador no se corresponde con ninguna codificación del protocolo, éste le pide al PC que le reenvíe la información.

Los 8 bits de una palabra disponibles en la comunicación serie, se han dividido de tal manera que cada operación a realizar por el microcontrolador se le asigna a una combinación de estos 8 bits. Con 8 bits se pueden codificar $2^8=256$ operaciones, pero en nuestro protocolo sólo se han utilizado 16. Así que se ha optado por agrupar los bits de dos en dos y dividir la palabra código en cuatro campos. De esta forma el microcontrolador puede extraer fácilmente los diferentes campos haciendo una simple AND con una máscara de unos en la posición adecuada. También se consigue que el código sea más robusto frente a los errores y borrones introducidos en el canal.

Tabla 11: Distribución de los 8 bits de la palabra enviada por el puerto serie.

Palabra código 8 bits (RS-232)							
STATUS		SWITCH		MODE		DIRECTION	
b₇	b₆	b₅	b₄	b₃	b₂	b₁	b₀
00 -		00 -		00 -		00 -	
01 - K.O.		01 - Stop Dw		01 - Half		01 - Left	
10 - OK.		10 - Stop Up		10 - Full		10 - Right	

A continuación se muestra una tabla con los valores codificados de las instrucciones de giro del motor que envía el PC hacia el microcontrolador.

Tabla 12: Codificación de las instrucciones de giro que envía el PC hacia el microcontrolador.

Instrucción giro PC		Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)							
Modo	Direccion			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
Full	Izquierda	9	0x09	0	0	0	0	1	0	0	1
Full	Derecha	10	0x0A	0	0	0	0	1	0	1	0
Half	Izquierda	5	0x05	0	0	0	0	0	1	0	1
Half	Derecha	6	0x06	0	0	0	0	0	1	1	0

Y el microcontrolador responde a estas instrucciones con el campo OK delante si se ha ejecutado correctamente, o con el KO si se ha pulsado el pulsador de paro de emergencia o STOP.

Tabla 13: Codificación de las respuestas a las órdenes de giro que envía el microcontrolador hacia el PC.

Respuesta giro microcontrolador			Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)							
Status	Modo	Dirección			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
OK	Full	Izquierda	137	0x89	1	0	0	0	1	0	0	1
OK	Full	Derecha	138	0x8A	1	0	0	0	1	0	1	0
OK	Half	Izquierda	133	0x85	1	0	0	0	0	1	0	1
OK	Half	Derecha	134	0x86	1	0	0	0	0	1	1	0
KO	Full	Izquierda	105	0x69	0	1	1	0	1	0	0	1
KO	Full	Derecha	90	0x5A	0	1	0	1	1	0	1	0
KO	Half	Izquierda	101	0x65	0	1	1	0	0	1	0	1
KO	Half	Derecha	86	0x56	0	1	0	1	0	1	1	0
Repetir Tx			80	0x50	0	1	0	1	0	0	0	0

El protocolo se ha diseñado para que en ningún caso el PC pueda seguir operando si el microcontrolador no ha respondido a la petición de giro o si no se han obtenido los datos del analizador de redes. Y si en algún momento se bloqueara el microcontrolador, gracias al *timeout* del puerto serie se devuelve el control al PC y se lanza un error.

Véase el código de la función en el *Anexo B*.

4.3.10. Finalizar la comunicación serie: *finalizar_puerto*

Esta función se encarga de finalizar la comunicación serie del PC con el microcontrolador y de liberar el puerto serie creado para este fin en el entorno Matlab. Antes de cerrar la comunicación se leen los datos pendientes en el buffer de entrada del puerto serie si fuere el caso. Posteriormente se cierra la comunicación con el envío de la orden codificada para tal efecto (ver *Tabla 14*).

Tabla 14: Codificación de la instrucción de finalizar la comunicación con el microcontrolador.

Instrucción de PC a microcontrolador	Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)									
			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀		
Finalizar comunicación	64	0x40	0	1	0	0	0	0	0	0		

Si el microcontrolador responde con la misma instrucción se concluye la comunicación satisfactoriamente, pero si no se recibe respuesta o no es la adecuada, se lanza un error y concluye la ejecución del programa principal (ver *Tabla 15*).

Tabla 15: Codificación de la instrucción de comunicación finalizada correctamente por el microcontrolador.

Instrucción de microcontrolador a PC	Codificación decimal	Codificación hexadecimal	Codificación binaria (RS-232)							
			b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
Comunicación finalizada	64	0x40	0	1	0	0	0	0	0	0

Véase el código de la función en el *Anexo B*.

4.4. Firmware

El firmware es el software programado en el microcontrolador y se encarga de traducir las instrucciones procedentes del PC en niveles de tensión que inciden sobre las entradas de los diferentes circuitos integrados del hardware y actúan, en última instancia, sobre las fases del motor a pasos.

En este apartado se explicará el entorno de programación utilizado, el lenguaje, el sistema de programación de la memoria del microcontrolador y los archivos componentes del firmware así como su funcionamiento global.

4.4.1. Entorno de programación: *MPLAB IDE v8.36* y *MPLAB ICD 2*

Como se ha escogido un microcontrolador de la marca *Microchip*, se ha utilizado el software de esta misma marca para programarlo. El software utilizado es el *MPLAB IDE* de *Microchip* en su versión v8.36 que se facilita gratuitamente en la web del fabricante.

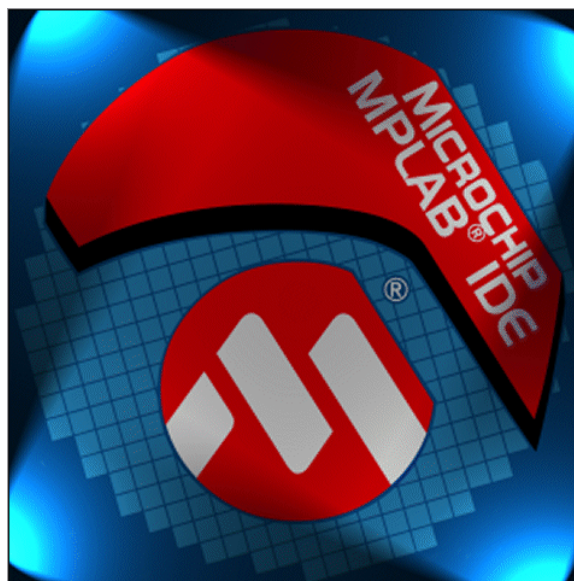


Fig. 56: Pantalla de inicio del programa *MPLAB IDE*.

Este programa ofrece un completo entorno tanto para programar y depurar el firmware como para insertarlo en la memoria del microcontrolador y debugarlo en tiempo real de ejecución o paso a paso. Como todo programa, primero debe crearse un *workspace* con las características principales del proyecto (microcontrolador utilizado, tipo de programación, lenguaje de programación, etc.). Posteriormente se crean o insertan los archivos que contienen el código fuente del firmware y las librerías utilizadas o archivos fuente. Seguidamente se compilan todos

los archivos y se traducen a lenguaje máquina para ser programados en la memoria del microcontrolador.

Para insertar el firmware en la memoria del microcontrolador se ha diseñado un circuito sobre el mismo hardware que permite la programación sin necesidad de sacar el microcontrolador de su zócalo (ver apartado 4.2.4 *Fuente de Vreg DC, MCLR e ICSP*). De esta manera se potencia la versatilidad del conjunto y se ahorra en circuitería y dispositivos de programación externos. A este tipo de programación se le denomina ICSP (*In-Circuit Serial Programming*) y ya es muy habitual encontrarla en numerosos microcontroladores.



Fig. 57: ICSP mediante MPLAB ICD 2.

Para poder realizar esta programación ICSP, es necesario disponer de un cable de conexión serie (RS-232) o de un dispositivo externo llamado MPLAB ICD 2 (*In-Circuit Debugger & Development*). En este proyecto se ha utilizado el ICD 2 ya que ofrece mayor versatilidad y velocidad en la transferencia de datos gracias a su conexión USB.



Fig. 58: MPLAB ICD 2 con conexión USB al PC y RJ-12 al hardware

Este dispositivo se conecta al PC mediante conexión USB y al hardware mediante conector RJ-12. Es parte del diseño hacer llegar las seis señales procedentes de este conector a los correspondientes pins del microcontrolador y adaptar, si es necesario, los niveles de tensión de las mismas en función del microcontrolador utilizado. También puede alimentar el hardware con 5V DC pero se aconseja que la alimentación sea externa para evitar un exceso de corriente por el dispositivo.

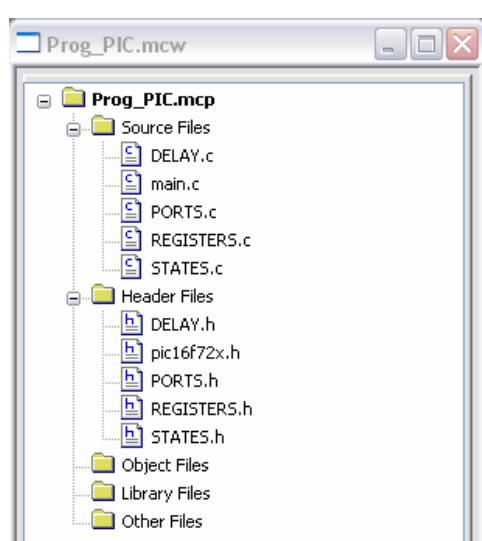
Al conectar el ICD 2 al hardware y abrir el proyecto en que se está trabajando, se establece una comunicación inmediata entre el programa y el microcontrolador. Esta funcionalidad es de gran utilidad ya que permite hacer tests al hardware y monitorizar las tensiones de alimentación y MCLR.

Destacar que existe una nueva versión llamada MPLAB ICD 3 con mayores funcionalidades pero no se ha podido tener acceso a ella y su coste es relativamente alto (alrededor de 200€).

4.4.2. Características del firmware

El lenguaje utilizado en la programación del código firmware ha sido lenguaje C. Es muy habitual que los microcontroladores sean programados en código *assembler*, pero esto obliga al programador a aprender todas las instrucciones que puede realizar el microcontrolador y sobretodo, las que no. Por esto y debido al gran conocimiento adquirido durante la carrera de lenguaje C, se ha decidido programar el firmware en este lenguaje, además de que al ser un lenguaje de mayor nivel que el *assembler*, se comprende mejor la estructura del código a simple vista.

Tras la decisión del lenguaje, se buscó un compilador de C compatible con la versión del programa para que tradujera el código C a un archivo tipo hexadecimal con las instrucciones máquina comprensible por la CPU del microcontrolador. Y más que un compilador se ha utilizado todo un entorno de herramientas que permiten el compilado desde el MPLAB IDE y que son compatibles con las familias PIC10/12/16 MCU de *Microchip*. El compilador elegido es el *HI-TECH C Tools for the PIC10/12/16 MCU Family*.



Una vez configurado el *workspace* del MPLAB IDE y el compilador, se empieza la elaboración del firmware. El código se ha dividido en diferentes archivos que se linkan tras la compilación. Los archivos se dividen en dos carpetas: *Source Files* para archivos con código fuente y *Header Files* para librerías y archivos que contienen las direcciones de memoria, nombre de los registros, etc.

A continuación se describe la funcionalidad de cada archivo y ya en los sucesivos apartados se comentan las particularidades de cada uno.

Fig. 59: Estructura jerárquica de los documentos del firmware.

- *DELAY.c*: este archivo contiene dos funciones que permiten retardar la ejecución del código mediante la repetición de instrucciones NOP.
- *main.c*: este archivo contiene el *thread* principal de ejecución. En él se inicializan las diferentes variables utilizadas y se invocan las funciones que inician los registros del microcontrolador. Posteriormente se entra en el bucle principal a la espera de órdenes. También es el encargado de atender las interrupciones mediante las ISR definidas en los registros.
- *PORTS.c*: en este archivo se configuran los 5 puertos del microcontrolador según sean entradas o salidas. También se habilitan las resistencias de *pull-up* y las interrupciones.
- *REGISTERS.c*: en este archivo se inicializan los registros del microcontrolador para la configuración específica del proyecto. Entre otros, se configura la frecuencia del oscilador interno, se habilitan las interrupciones de los puertos y se configura la transmisión serie.
- *STATES.c*: este archivo contiene las funciones que actualizan el estado en que se encuentran las fases del motor y activan las salidas necesarias para alimentarlas. En el apartado ‘4.4.8 Activar las fases del motor: *STATES.c*’ se explicará con más detalle el funcionamiento de cada una de las funciones integradas en este archivo.
- *DELAY.h*: en este archivo se definen las llamadas externas de las dos funciones de retardo programadas en el archivo *DELAY.c*.
- *pic16F72x.h*: archivo que contiene el mapeado de memoria y la asignación de las direcciones de memoria de los registros de los microcontrolador de la familia 16F72x.
- *PORTS.h*: en este archivo se renombran los pins del microcontrolador asociándolos con los nombres de las variables del código para facilitar la programación.
- *REGISTERS.h*: en este archivo se define la llamada externa de la función programada en el archivo *REGISTERS.c*.
- *STATES.h*: en este archivo se definen las llamadas externas de las funciones programadas en el archivo *STATES.c*.

Una vez que se compila el código y se verifica que no hay errores se inicia la programación del mismo en la memoria del microcontrolador. Antes de iniciar la programación, debe verificarse que el hardware esté alimentado, que la conexión con el dispositivo MPLAB ICD 2 sea correcta y que la memoria del microcontrolador esté vacía. Una vez verificado lo anterior, se procede a la programación del firmware sobre la memoria del microcontrolador.

Al finalizar la programación, aparece un mensaje de verificación o de error en la línea de comandos del MPLAB IDE. Si todo ha sido correcto, el microcontrolador ya contiene el firmware y está listo para ejecutarse cuando se deshabilite el MCLR.

Posteriormente puede utilizarse el modo de depuración de errores para debugar el firmware desde el PC mientras se ejecuta en el microcontrolador. Esta herramienta es de gran utilidad para evitar puntos muertos (o *dead points*) o averiguar en que punto se encuentra el código y, como el firmware se ejecuta directamente sobre el hardware, pueden activarse los diferentes pulsadores.

4.4.3. Hilo principal del firmware: *main.c*

main.c es el corazón del firmware y en él se halla el bucle principal de ejecución. Después de los *includes* se encuentra la definición de las variables y posteriormente el *main*. Como primer apartado del *main* se encuentra la inicialización de los puertos, los registros y los estados. Seguidamente el bucle principal con el '*while(1)*' y ya dentro de él la condición de que la comunicación serie esté correctamente abierta (*COMMS_OK*).

Una vez establecida la comunicación serie con el PC, se esperan las órdenes para realizar los giros con la variable *step*, que se activa sólo cuando se recibe una orden de giro. Esta variable se activa en la ISR de lectura del puerto serie después de verificar que se trata de una instrucción únicamente de giro. Entonces se llama a la función *PUSH_STATE* que se encarga de actualizar el estado y hacer girar el motor. Una vez finalizado el giro, se desactiva la variable *step* y se desactivan las fases del motor para no consumir energía y no saturar los transistores de potencia.

La segunda parte y no menos importante del archivo son las rutinas de atención a las interrupciones o ISR (*Interrupt Service Routine*). Al activar las interrupciones de ciertos pins de los puertos del microcontrolador debe de hacerse una gestión de las interrupciones, y es en la función *interrupt ISR* en donde se gestionan estas interrupciones.

Primero se encuentra la gestión de las interrupciones del puerto C (*RCIF*) que se corresponde con la entrada de un byte en el puerto serie o RS-232. Se lee el byte recibido y se ejecuta la instrucción recibida en función de los valores de las variables del hardware. Hay tres tipos de instrucciones: abrir comunicación, cerrar comunicación y realizar giro.

Dentro de las dos primeras se responde con distintas acciones dependiendo de si la comunicación ya está establecida o si ya está cerrada. En el caso de que la instrucción sea girar el motor, se extrae el modo y la dirección de giro y se activa la variable *step* para realizar el giro.

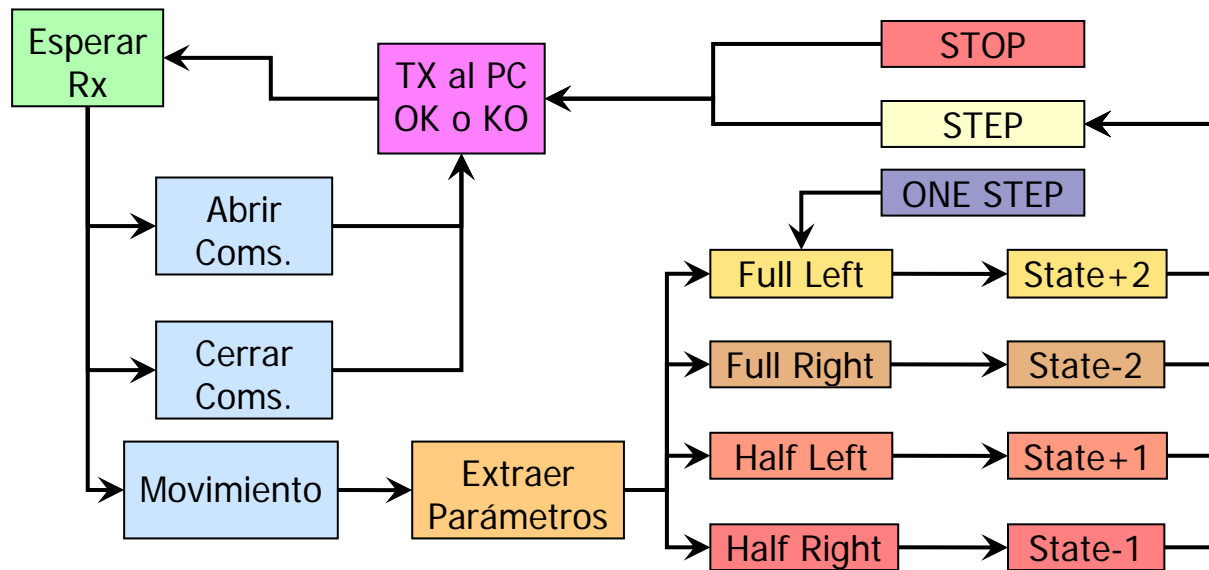


Fig. 60: Diagrama de flujo de las principales instrucciones del firmware.

A continuación se encuentra la gestión de las interrupciones del puerto B (*RBIF*) que se corresponden con los dos *switch* del hardware. En un principio estas interrupciones tenían que ser activadas por dos finales de carrera de tal manera que el motor no pudiera girar en el sentido que estuviera activo el final de carrera. Finalmente se sustituyeron los finales de carrera por dos pulsadores tipo N.A. que activan las interrupciones al ser pulsados.

Un pulsador, el de palanca, se encarga de realizar un paso a la izquierda cuando se pulsa siempre y cuando no se esté en medio de una sintonización. La rutina de atención a la interrupción es la encargada de llamar a la función *PUSH_ONE_STEP()* que realiza el paso.

El otro pulsador, el redondo, es el paro de emergencia o STOP y sirve para detener el autoajuste del filtro en cualquier momento.

Al final de las rutinas del puerto B, tenemos que resetear el *flag* para recibir nuevas interrupciones. Esto no ocurre con el puerto C ya que inmediatamente después de leer el byte del puerto serie se resetea automáticamente el *flag* de este puerto.

Véase el código de la función en el *Anexo C*.

4.4.4. Configuración de los puertos del microcontrolador: *PORTS.c*

Dentro de este archivo se encuentra la función *INIT_PORTS()* que se encarga de configurar los cinco puertos del microcontrolador. Primeramente se configura el puerto A como digital y se habilitan sus cuatro salidas RA1, RA2, RA3 y RA4.

Seguidamente se configura el puerto B también como digital. Las entradas RB0 y RB1 se corresponden con los pulsadores del hardware y como deben generar interrupciones, se habilitan las resistencias de *pull-up* y las interrupciones de estos dos pins.

El puerto C se configura automáticamente cuando se activa la comunicación serie RS-232 y lo único que hacemos es habilitarlo. Sólo comentar que el microcontrolador recibe los datos serie por el pin 26 (RC7) y transmite por el pin 25 (RC6).

Para no dejar ningún pin ‘al aire’, se configura el puerto D como digital y se habilita, pero al no concretar si los pins son entradas o salidas, se quedan en un estado de alta impedancia.

Finalmente el puerto E se configura como digital y se habilitan sus tres salidas RE0, RE1 y RE2 que se corresponden con los tres leds verdes de información.

Véase el código de la función en el *Anexo C*.

4.4.5. Header de los puertos: *PORTS.h*

En este archivo se definen los nombres de los pins de los puertos para facilitar la programación del firmware y poder deducir del mismo nombre la funcionalidad que tiene dicho pin. También se define la llamada de las funciones del archivo *PUERTO.c*.

Véase el código de la función en el *Anexo C*.

4.4.6. Configuración de los registros: *REGISTERS.c*

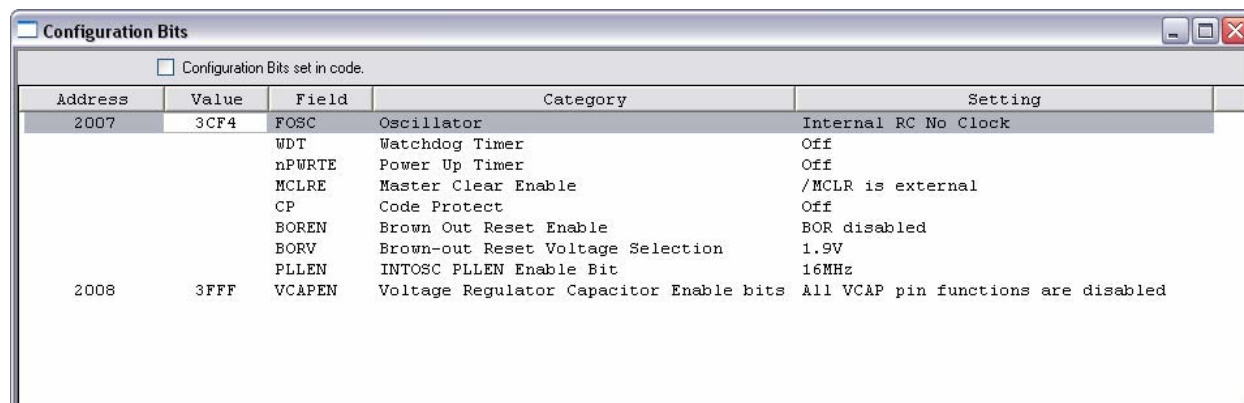
En este archivo se encuentra la función *INIT_REGS()* que configura los registros que no pertenecen a los puertos y los que no se pueden configurar directamente desde el PC. Principalmente se configuran tres tipos de registros: los registros de la frecuencia de oscilación, los registros que habilitan las interrupciones y los registros que configuran la comunicación serie.

Como se dispone de tres bits para elegir la frecuencia de oscilación, se pueden seleccionar 8 frecuencias del reloj interno comprendidas entre 62.5kHz y 16MHz. En este proyecto se ha configurado el reloj interno a su máxima frecuencia, 16MHz.

También se han habilitado las interrupciones generales de todo el microcontrolador y en concreto las que proceden de periféricos. Cabe destacar que no solamente existen este tipo de interrupciones, sino que también hay interrupciones de *timers* internos.

Para finalizar, se configuran los registros de la comunicación serie RS-232 con la misma configuración que el Matlab y se habilita la recepción y transmisión.

Hay unos registros que pueden configurarse directamente desde el programa MPLAB IDE accediendo a *Configure* → *Configuration Bits*. Estos registros pertenecen a dos palabras de configuración que siempre deben configurarse. A continuación se muestra la configuración de estos registros para este proyecto.



Address	Value	Field	Category	Setting
2007	3CF4	FOSC	Oscillator	Internal RC No Clock
		WDT	Watchdog Timer	Off
		nPWRT	Power Up Timer	Off
		MCLR	Master Clear Enable	/MCLR is external
		CP	Code Protect	Off
		BOREN	Brown Out Reset Enable	BOR disabled
		BORV	Brown-out Reset Voltage Selection	1.9V
		PLLEN	INTOSC PLL EN Enable Bit	16MHz
2008	3FFF	VCAPEN	Voltage Regulator Capacitor Enable bits	All VCAP pin functions are disabled

Fig. 61: Ventana con la configuración de los registros configurables desde el MPLAB IDE.

Véase el código de la función en el *Anexo C*.

4.4.7. Header de los registros: *REGISTERS.h*

En este archivo únicamente se define la llamada de la función *INIT_REGS()* definida en el archivo *REGISTERS.c*.

Véase el código de la función en el *Anexo C*.

4.4.8. Activar las fases del motor: *STATES.c*

Este archivo es el segundo en importancia después del *main* ya que en él se actualiza el estado en el que se encuentra el firmware y, dependiendo del estado, se activan unas fases del motor u otras. Este archivo contiene dos funciones principales: *PUSH_STATE()* y *PUSH_OUTPUT()*.

PUSH_STATE() es la función que actualiza el estado dependiendo del modo y dirección de giro del motor. Como el motor a pasos dispone de cuatro fases y se puede realizar medio paso entre fase y fase, hay un total de ocho estados posibles. Es por este motivo que se han creado ocho estados y dentro de cada estado se activan diferentes fases (ver *Fig. 62*).

El funcionamiento es sencillo; existe una variable denominada *STATE* que indica el estado actual en el que se encuentran las fases del motor. Si se recibe la orden desde el PC de girar medio paso a la izquierda, la variable *STATE* se incrementa en 1. Si se recibe la orden de girar medio paso a la derecha se decrementa en 1. Pero si en lugar de medio paso se requiere un paso entero, la variable se incrementa o decrementa en 2, dependiendo del sentido de giro deseado.

Evidentemente se ha creado una estructura modular de 8 estados y siempre se rota entre los estados 0 y 7 ya que se hace el módulo base 8 de la variable *STATE*.

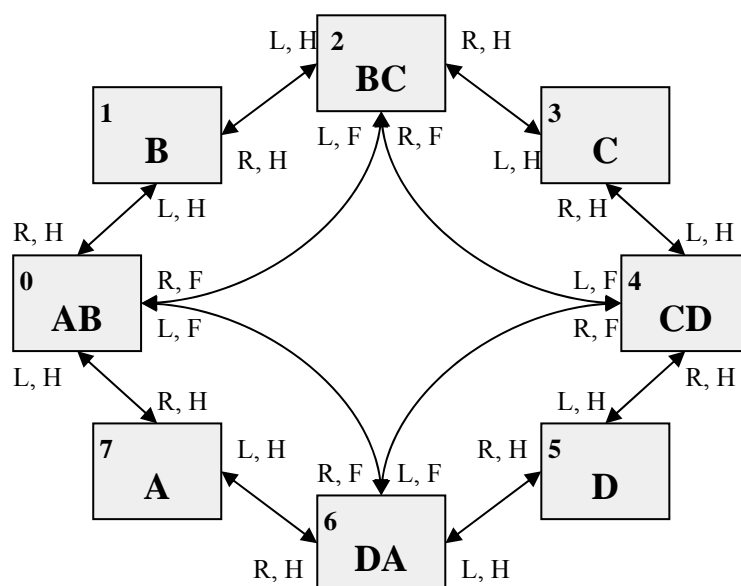


Fig. 62: Diagrama de estados de las fases activas del motor.

Como se puede ver en la *Fig. 62*, hay ocho estados posibles enumerados del 0 al 7 en la parte superior izquierda. En el centro se muestra la fase o fases activas en dicho estado (A, B, C o D). Los conectores muestran las condiciones que deben cumplirse para pasar de un estado a otro. Estas condiciones hacen referencia a las variables *Dirección* y *Modo*. La variable *Dirección* puede tomar los valores L o R (del inglés *Left* o *Right*) haciendo referencia a la dirección de giro del motor. Y la variable *Modo* puede tomar los valores H o F haciendo referencia al tipo de paso si es medio o entero (del inglés *Half* o *Full*).

Por ejemplo, si el valor actual de la variable *STATE* es 2, las fases activas son la B y la C. Seguidamente recibimos la orden de girar medio paso a la izquierda con lo cual el valor la variable *Modo* es *Half* (H) y el de la variable *Dirección* es *Left* (L). Así que el valor de *STATE* pasa a ser 3 y se activa únicamente la fase C.

Una vez actualizado el estado, se dispone a actuar sobre los pins del microcontrolador para activar las fases del motor mediante la función *PUSH_OUTPUT()*.

PUSH_OUTPUT() es la función que activa los pins del microcontrolador que, tras pasar por la etapa lógica, aislamiento y potencia, activan las fases del motor. Se le pasa como parámetro la variable *STATE* y dependiendo de ésta se activan unos pins u otros durante cierto tiempo para permitir el giro del motor. Finalizado este tiempo se desactivan las fases para reducir el consumo de energía ya que no es necesario un par estático de retención.

Si el movimiento se ha efectuado correctamente se confirma al PC mediante la instrucción codificada correspondiente. Si la orden no se ha ejecutado correctamente se envía un mensaje de error al PC. Y si no se ha entendido el movimiento a efectuar, se le pide al PC que reenvíe los datos.

Dentro de este archivo también hay dos funciones más. La función *INIT_STATE()* es la encargada de inicializar la variable *STATE* a cero. Sólo se ejecuta al inicio del *main* y de esta forma, si se realizan diferentes sintonizados encadenados, se conserva el estado final del sintonizado anterior evitando movimientos bruscos.

Y la función *PUSH_ONE_STEP()* es la que ejecuta un único paso a la izquierda cuando se pulsa el *switch* de palanca y no se está realizando ningún sintonizado. Si también se pulsa el *switch* de *STOP* juntamente con el de un paso, no se realiza el paso a la izquierda y el led D6 parpadea 5 veces.

Véase el código de la función en el *Anexo C*.

4.4.9. Header de los estados: *STATES.h*

En este archivo únicamente se define la llamada de las funciones definidas en el archivo *STATES.c*. Estas funciones son: *INIT_STATE()*, *PUSH_ONE_STEP()*, *PUSH_STATE()*, *PUSH_OUTPUT()*.

Véase el código de la función en el *Anexo C*.

4.4.10. Funciones de retardo: *DELAY.c*

Este archivo contiene dos funciones para retardar la ejecución del código del microcontrolador sin necesidad de dormirlo ni activar ningún tipo de *timer* con su correspondiente gestión de la interrupción. Se trata de realizar repetidamente la función '*nop*' o negación. Esta función se realiza con dos ciclos de reloj y es por este motivo que su ejecución dura 125ns en lugar de los 62.5ns correspondientes al período de 16MHz.

Una vez hecha la equivalencia del número de operaciones '*nop*' a realizar para lograr 1µs, que son ocho, solamente hace falta multiplicar el retardo en microsegundos deseado por el número de veces que tiene que realizarse la operación '*nop*', o sea, por ocho.

Véase el código de la función en el *Anexo C*.

4.4.11. Header de los retardos: *DELAY.h*

En este archivo únicamente se definen las llamadas de las funciones *DELAY_US()* para retardar microsegundos o *DELAY_MS()* para retardar milisegundos. Estas dos funciones están definidas en el archivo *DELAY.c*.

Véase el código de la función en el *Anexo C*.

4.4.12. Mapeado de memoria de los registros del microcontrolador: *pic16f72x.h*

En este archivo se asignan los nombres de los registros a las direcciones de memoria del microcontrolador designadas para tal efecto, es decir, que no son direcciones de datos del programa. En él se define cada uno de los bits de los registros disponibles en el microcontrolador y que luego pueden ser accesibles por el programa para leer o modificar su valor únicamente introduciendo su nombre. De esta forma nos ahorramos tener que recordar la dirección de memoria del bit MCLREN, por ejemplo.

El funcionamiento del archivo es sencillo; al principio se asignan las direcciones de inicio de las palabras registro. Como cada palabra consta de 8 bits, posteriormente y para definir la dirección de un bit concreto de una palabra, se referencia a la dirección inicial de la palabra multiplicada por 8 y más el *offset* correspondiente al bit en cuestión.

Por ejemplo el registro *carry*. Este registro se encuentra dentro de la palabra de registros *STATUS* en la posición de bit cero. Primero se ha definido la dirección inicial de la palabra *STATUS* que se corresponde con la dirección número 3 (@ 0x003). Posteriormente se define el registro *carry* como la dirección número 3 multiplicada por ocho (para dar cabida a los 8 bits de la palabra) más 0 (que es el *offset* dentro de la palabra del registro *carry*). Como resultado obtenemos la dirección 24 (@ 0x018).

Véase cómo se implementa en el archivo.

```
volatile unsigned char STATUS      @ 0x003;
volatile bit          CARRY        @ ((unsigned)&STATUS*8)+0;
```

De esta forma muy ingeniosa se consigue fragmentar la memoria del microcontrolador y estructurarla en palabras de registros de 8 bits (multiplicando por 8) y añadiendo un *offset* para indicar de qué bit se trata.

Véase el código de la función en el *Anexo C*.

4.5. Estructura de soporte y mecanizado

Como último paso en la implementación del proyecto sólo hacía falta construir una estructura de soporte para el motor y el hardware. De esta forma se permite la alineación del motor con el tornillo de sintonización del resonador de microondas utilizado. También se tuvo que ultimar algunos detalles como el encaje de la puntera del motor y la selección del tornillo de sintonización del resonador.

En la *Fig. 63* a continuación, se puede ver una imagen del resultado final del proyecto que incluye la estructura de soporte y el hardware diseñados, juntamente con el motor a pasos.

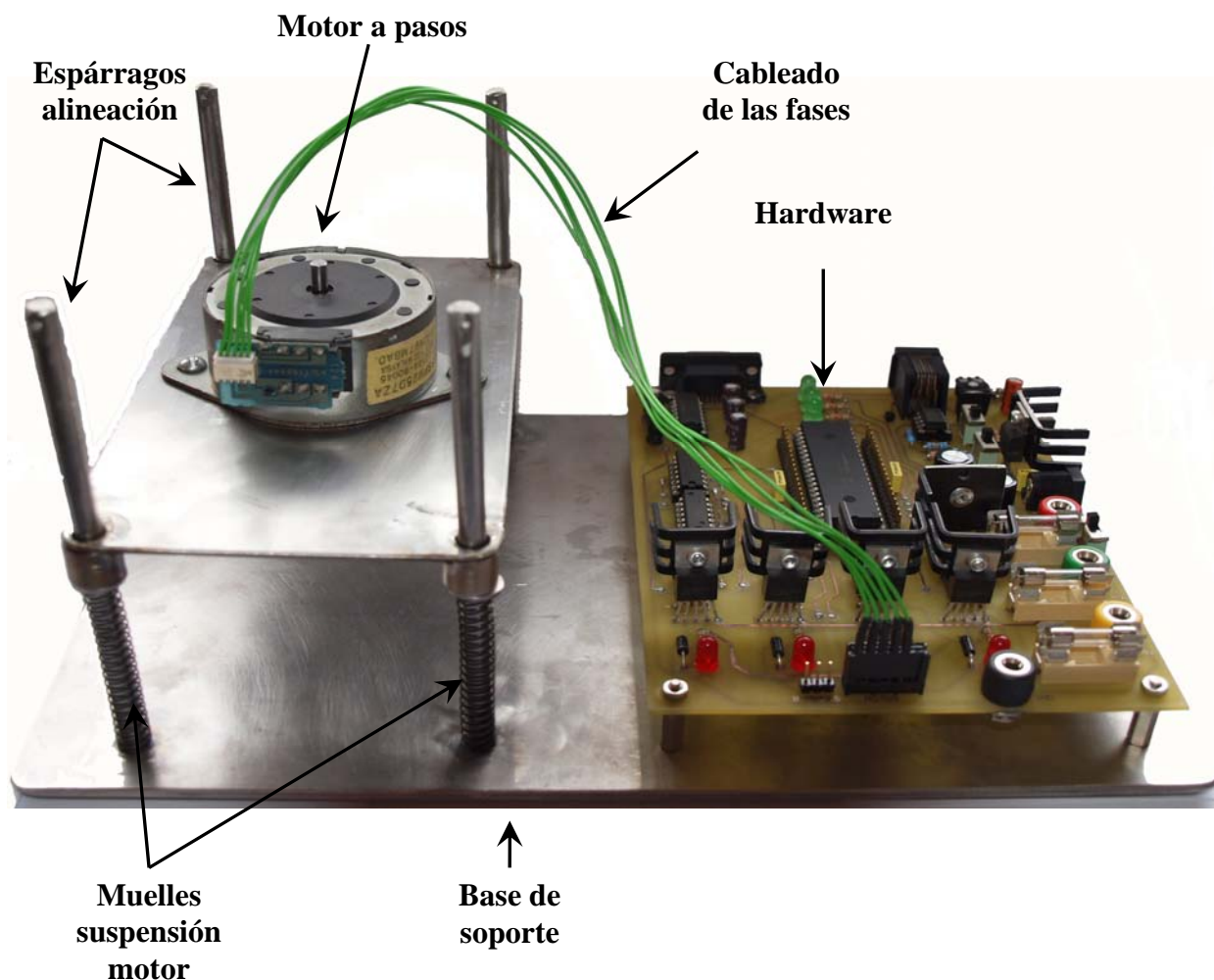


Fig. 63: Estructura de soporte con el hardware y el motor.

En los siguientes apartados se comentan los problemas ‘mecánicos’ más destacados que se tuvieron durante la realización de la estructura de soporte y mecanizado y cómo se solucionaron.

4.5.1. Estructura de soporte

En la estructura de soporte el único inconveniente fue el peso del motor y su ubicación justo encima del resonador de microondas. Para ello, se ingenió un sistema de suspensión del motor mediante una plancha metálica guiada por cuatro espárragos metálicos en cada esquina. Para solucionar el problema del peso del motor, se optó por introducir unos muelles dentro de los espárragos a modo de amortiguación. Estos muelles han sido ajustados a la medida del resonador utilizado.



Fig. 64: Estructura de soporte del motor a pasos y el hardware.

Como se ve en la *Fig. 64*, la estructura consta de una base rectangular en donde se han soldado verticalmente unos espárragos que hacen de guía para alinear la plancha que soporta el motor. Dentro de los espárragos se han ubicado unos muelles con arandelas en la parte superior encargados de suspender el motor a la altura idónea para que el peso del motor no recaiga en exceso sobre el tornillo de sintonización. Se ha de tener en cuenta que el desplazamiento máximo del tornillo de sintonización es de 1cm y por lo tanto los muelles tienen suficiente recorrido debido a la flexión ejercida por el peso del motor.

En un principio no se introdujeron los muelles pero rápidamente se vio que la fuerza ejercida sobre el tornillo de sintonización del resonador era excesiva y el roce de la plancha que soporta el motor con los espárragos hacía que no subiera de forma uniforme debido a que el peso no estaba exactamente equilibrado ya que se centraba todo el apoyo en la puntera del motor. Para solucionar este problema se introdujeron los cuatro muelles. De esta forma se solucionaba el problema del peso del motor y se conseguía descentralizar la fuerza ascendente que compensa el peso del motor hacia las cuatro esquinas.

También se hicieron pruebas con muelles en la parte superior e inferior de la plancha que sostiene el motor de tal manera que la plancha quedaba suspendida a cierta altura, pero se desestimó porque la fuerza ejercida por los muelles superiores hacia el resonador era excesiva y el recorrido mínimo.

El material con que está realizado el soporte y los espárragos es acero inoxidable pulido. Los espárragos están soldados con soldadura eléctrica de puntos sobre la misma base y se han lubricado con grasa para reducir la fricción con la plancha que soporta el motor. En el dorso se han ubicado cuatro patas de silicona para no dañar la mesa en que se sitúe.

Véase en la *Tabla 16* a continuación las características físicas del soporte.

Tabla 16: Características físicas de la estructura de soporte.

Dimensiones de la base	Dimensiones de los espárragos
Largo x Ancho x Alto: 270 x 175 x 3 (mm.)	Alto x Diámetro: 120 x 6 (mm.)
Dimensiones base motor	Peso (sin resonador)
Largo x Ancho x Alto: 150 x 90 x 2 (mm.)	1.980 (gr.)
Material	
Acero inoxidable pulido	

4.5.2. Puntera del motor

Un reto importante fue el acople de una puntera multiusos al eje del motor a pasos. Para dotar de mayor versatilidad al conjunto y poder sintonizar diferentes filtros con diferentes tornillos de sintonización, se decidió proveer al motor de una puntera multiusos con un enclave Allen de 6mm estándar. De esta forma se pueden intercambiar las punteras de tipo estándar sin necesidad de cambiar el anclaje.

Se partió de una puntera multiusos con anclaje Allen de 6mm y se rectificó manualmente en un torno para practicarle un agujero en el centro que encajara con el eje del rotor del motor a pasos. Para asegurar que no se mueve de su sitio se le ha insertado un tornillo prisionero muy pequeño que fija la puntera al eje del motor. La puntera multiusos dispone de un imán en su interior que atrae las puntas insertadas y evitar así que caigan ya que se encuentran en posición vertical.

En la *Fig. 65* se puede ver la puntera multiusos con una punta Allen intercambiable. También se aprecia el tornillo prisionero que fija la puntera al eje del rotor.



Fig. 65: Detalle de la puntera multiusos encajada en el eje del rotor.

4.5.3. Tornillo de sintonización

Normalmente en los filtros sintonizables se utilizan tornillos de tipo plano ya que como su ajuste es manual se facilita mucho la tarea al no requerir herramientas específicas. En casi todos los laboratorios se puede disponer de un destornillador plano.

Sin embargo, en las pruebas realizadas tras la finalización de la estructura, se observó que si el tornillo de sintonización del resonador y la puntera del eje del motor no estaban perfectamente alineados, transcurridos unos pasos, la puntera se desencajaba de la ranura plana del tornillo. Y como algunos filtros suelen haber varios tornillos de sintonización, se podría colisionar con los tornillos contiguos.

Éste fenómeno también se veía agravado debido al desequilibrio de la plancha que sujeta el motor ya que es muy difícil equilibrar todo el peso del motor en su eje. Y debido a la complejidad de mecanización de la puntera y la presión del tornillo prisionero que la sujeta al eje, la puntera también tiene una pequeña deriva hacia un lado que acentúa aún más este fenómeno.

Así que se decidió cambiar el tornillo de sintonización plano por uno con anclaje Allen. De esta forma, al enclavar la puntera dentro de la cabeza del tornillo se evita que se salga y pueda colisionar con los tornillos contiguos. Además en las pruebas realizadas se ha puesto una espuma protectora en la parte inferior de la estructura para que el resonador pueda inclinarse un poco ante la fuerza de giro del motor, y como el motor está suspendido entre los cuatro espárragos de alineación, el motor también puede inclinarse y compensar el desajuste de su centro de gravedad.

El tornillo seleccionado es un tornillo de rosca cilíndrica para uniones metálicas con rosca métrica de 5mm estándar, cabeza Allen de 4mm, longitud de 6mm y tratamiento pavonado¹ anticorrosión.



Fig. 66: Tornillo de sintonización tipo Allen métrico 5mm.

¹ El **pavonado** consiste en la aplicación de una capa superficial de óxido abrillantado, compuesto principalmente por óxido férrico (Fe_2O_3) de color azulado, negro o café, con el que se cubren las piezas de acero para mejorar su aspecto y evitar su corrosión.

5. Resultados experimentales

Tras la implementación de las cuatro partes esenciales del proyecto (software, firmware, hardware y estructura de soporte) se procede a la realización de las primeras pruebas. Cabe destacar que en estas pruebas en ningún caso se pretende hacer un análisis exhaustivo del resonador utilizado o de sus parámetros, sino verificar el correcto funcionamiento del sistema diseñado para la sintonización de la frecuencia central y la presentación de los resultados para el análisis posterior.

En el inicio se ejecutaron unas pruebas sin el analizador de redes haciendo una simulación de la frecuencia central del resonador mediante un vector de frecuencias equidistantes dentro del rango de medida. Esta simulación permitió realizar pruebas sin disponer de un analizador de redes y comprobar el correcto funcionamiento tanto del hardware como del software sin el riesgo de dañar el resonador u otros componentes externos.

Con estas pruebas iniciales se ajustaron algunos parámetros, se retocó el software y el firmware y se ultimaron los detalles para las pruebas, ahora sí, con el analizador de redes.

La finalidad del proyecto es sintonizar un resonador de microondas automáticamente y el primer objetivo a alcanzar para lograr tal fin es medir la frecuencia central del resonador. Este objetivo se ve cumplido ya que la medida del analizador de redes es interpretada por la función correspondiente en el software y tras un pequeño cálculo se obtiene la frecuencia central del resonador.

Luego el software toma la decisión de subir o bajar la frecuencia para sintonizar el resonador a la frecuencia deseada. Este objetivo también se cumple ya que si la frecuencia actual del resonador es inferior a la deseada, el software toma la decisión de girar el motor en sentido antihorario, y si es superior en sentido horario.

El siguiente objetivo a cumplir es la sintonización lo más fina posible de la frecuencia. Esto se consigue gracias al medio paso que realiza el motor tras alcanzar la frecuencia deseada y evaluar la mínima diferencia entre todas las frecuencias y sintonizar la más cercana a la deseada.

Y el último objetivo se consigue cuando se muestran por pantalla los resultados de la sintonización en una gráfica de la evolución de la frecuencia central del resonador sintonizada en cada paso, el incremento de frecuencia entre pasos consecutivos y las pérdidas de inserción.

A continuación se muestran los resultados obtenidos en diferentes sintonizaciones y parametrizaciones del resonador.

5.1. Parametrización

5.1.1. Procedimiento y descripción de la parametrización

Para definir los límites superior e inferior de las frecuencias sintonizables por el resonador, se ejecuta el software en modo parametrización. Se sitúa el resonador en la posición de medida conectado al analizador de redes y con el tornillo de sintonización lo más introducido posible dentro del agujero pero cerca del rango en donde empieza a aumentar la frecuencia sintonizada. Para saber este punto en donde situar el tornillo, es importante tener conocimiento del rango de frecuencias útiles y haber visualizado anteriormente la respuesta del resonador en el analizador de redes.

El resonador utilizado dispone un único tornillo de sintonización que hace variar la frecuencia central. El rango de frecuencias sintonizables comprende desde los 5,9GHz hasta los 8,9GHz con unas pérdidas de inserción de -8dB aproximadamente.

El rango de medida del analizador de redes se fija entre 5GHz y 10GHz, y se han elegido 200 puntos de medida repartidos uniformemente entre este rango ya que el tiempo de medida no es excesivo (1seg.) y la precisión es muy buena (25MHz). Si se aumenta el número de puntos, la medida se ralentiza y pasa a tener valores muy altos llegando a tardar hasta 5seg. en el analizador utilizado.

Configurado el analizador de redes y situado el resonador en su posición de medida, solamente hace falta llamar la función que inicia todo el proceso de parametrización (*main_sintonizacion*) e introducir los parámetros necesarios requeridos por el software de control, y ya finalmente se inicia el proceso de parametrización del resonador.

Este proceso consiste en recorrer paso a paso todo el rango de medida del analizador de redes calculando la frecuencia central del resonador a cada paso y representar de manera gráfica los resultados obtenidos. Así, se obtiene una gráfica con las frecuencias sintonizables por el resonador y lo que es más importante, la frecuencia máxima y mínima a la que se puede centrar. Estas variables son importantes para no dañar el resonador intentando sintonizar frecuencias inferiores a las que puede alcanzar, ya que para sintonizarlas se debe introducir el tornillo dentro de la cavidad y llegar a dañarla si se introduce en exceso.

Una vez terminada la parametrización se presentan los resultados expuestos en el siguiente apartado.

5.1.2. Resultados de la parametrización

Una vez terminada la parametrización del resonador, se presentan los resultados de manera gráfica y se guardan los parámetros más importantes en un archivo específico para cada filtro. Estos parámetros son la frecuencia mínima y máxima sintonizable por el filtro parametrizado.

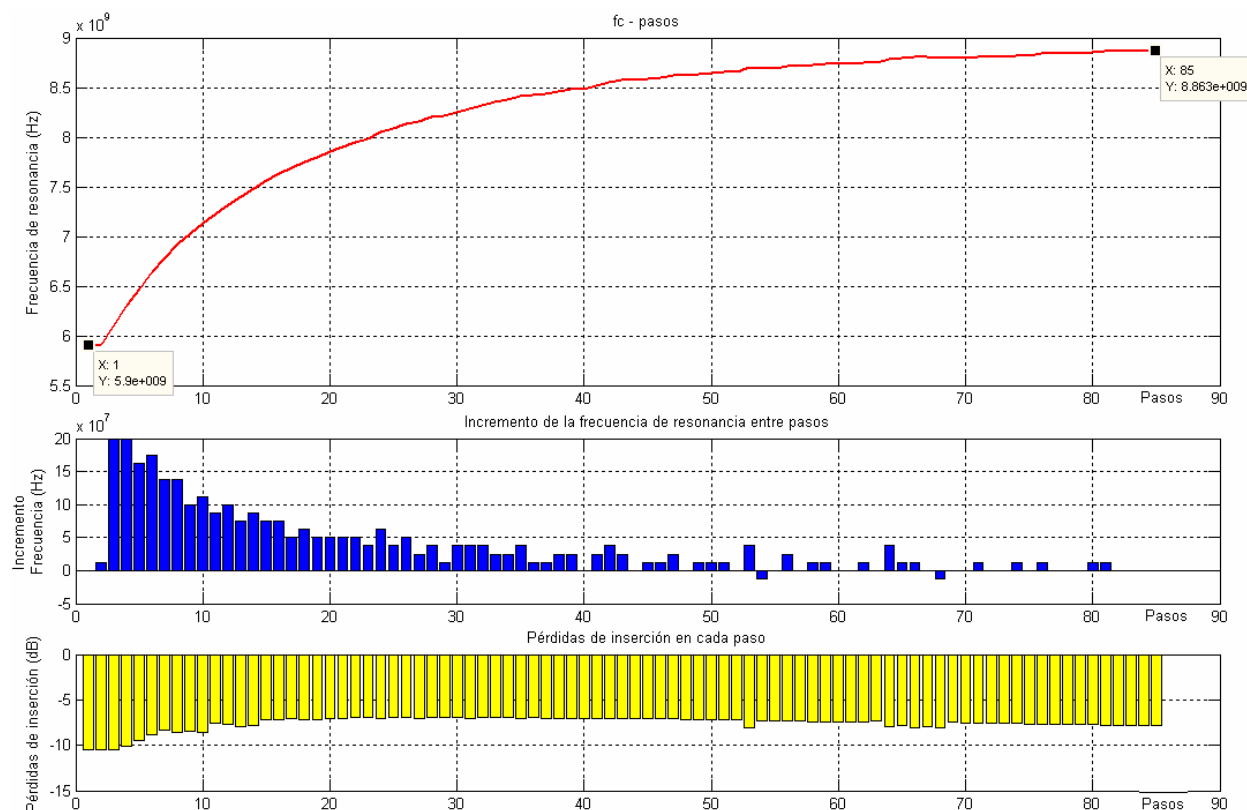


Fig. 67: Resultados de la parametrización del resonador con un rango de frecuencias sintonizables de 5,9GHz a 8,9GHz, y un rango de medida del AR de 5GHz a 10GHz con 200 puntos.

En la Fig. 67 se observa el resultado gráfico de la parametrización de un resonador con un rango de frecuencias sintonizables de 5,9GHz a 8,9GHz. Se alcanza su frecuencia máxima con 81 pasos, que corresponden a 1,7 vueltas del tornillo de sintonización. Realmente sorprende que solamente con menos de dos vueltas se pase del valor mínimo al máximo. Pero debido a la construcción del resonador, solamente hay 3mm entre la tapa superior que sujeta el tornillo y la plancha intermedia que contiene el resonador, siendo éste el único espacio que el tornillo puede penetrar. Pero este mismo hecho justifica la necesidad de instrumentos de ajuste automáticos y de gran precisión como el desarrollado en este proyecto.

En las Fig. 68 a Fig. 73 se muestran las pérdidas de inserción (S_{21}) del resonador para diferentes pasos del motor durante la parametrización. Como puede verse, el pico de la frecuencia central es distinguible en cualquiera de las imágenes e incluso a frecuencias inferiores en donde las pérdidas son mayores. Se observa claramente como el pico se desplaza hacia la derecha a medida que el tornillo de sintonización sale de la cavidad resonante.

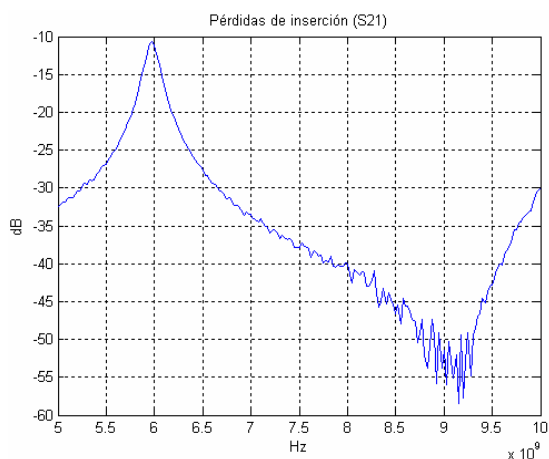


Fig. 68: Pérdidas de inserción en el paso 2 con $f_c=5,9625\text{GHz}$.

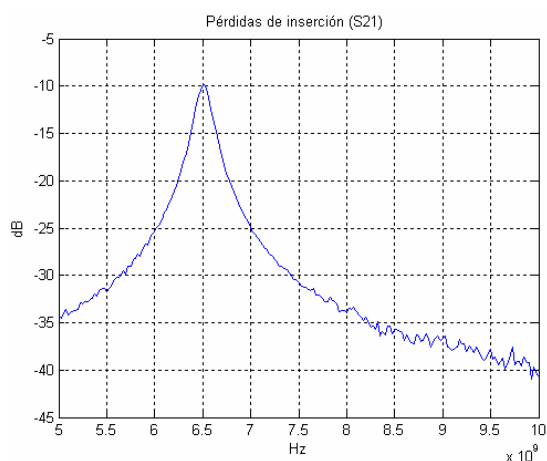


Fig. 69: Pérdidas de inserción en el paso 5 con $f_c=6,5\text{GHz}$.

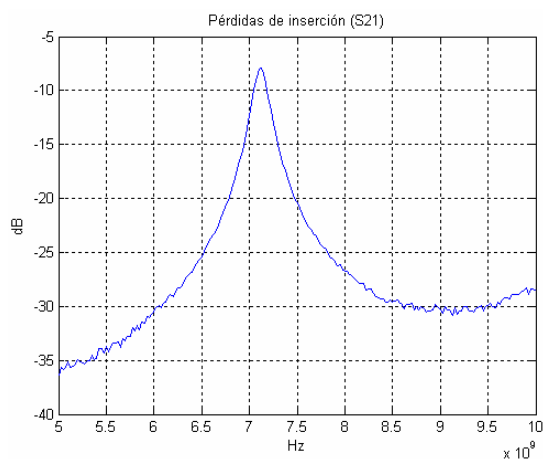


Fig. 70: Pérdidas de inserción en el paso 10 con $f_c=7,125\text{GHz}$.

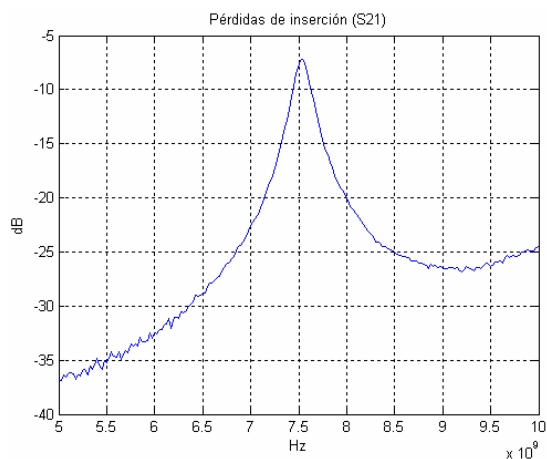


Fig. 71: Pérdidas de inserción en el paso 15 con $f_c=7,5375\text{GHz}$.

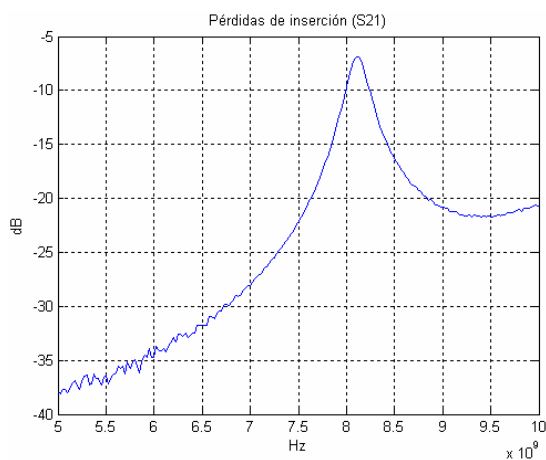


Fig. 72: Pérdidas de inserción en el paso 26 con $f_c=8,1125\text{GHz}$.

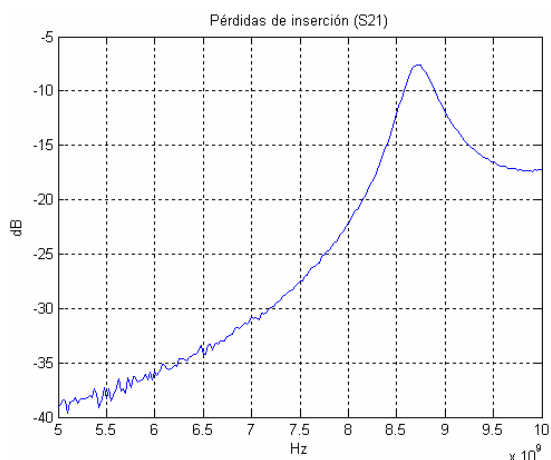


Fig. 73: Pérdidas de inserción en el paso 62 con $f_c=8,7375\text{GHz}$.

5.1.3. Análisis de los resultados

La primera gráfica de la *Fig. 67* (fc - pasos) se corresponde con la frecuencia del resonador sintonizada en cada paso. Se observa que el rango de frecuencias sintonizables por el resonador empieza a 5,9GHz y termina a 8,863GHz. Además se puede observar claramente que la variación de frecuencia es mucho mayor a frecuencias bajas. Concretamente el 90 por ciento de las frecuencias sintonizables se hallan en la primera mitad de los pasos (los 45 primeros pasos) y en la segunda mitad la frecuencia varía solamente 1GHz.

Así y según los resultados obtenidos en el párrafo anterior, a priori nos resultaría más difícil sintonizar de forma precisa una frecuencia baja que una frecuencia alta dentro del rango sintonizable, y en los siguientes resultados se pondrá de manifiesto esta afirmación.

También se corrobora el correcto funcionamiento de la función que parametriza el resonador ya que se ha establecido un número máximo de seis pasos con la misma frecuencia para detectar el límite superior, y como puede verse, en el paso 86 termina la ejecución tras seis pasos con la misma frecuencia.

En la segunda gráfica de la *Fig. 67* (incremento de la frecuencia de resonancia entre pasos) se muestra el incremento o decremento de la frecuencia de resonancia entre pasos consecutivos. Se aprecia claramente que en los primeros pasos el incremento de la frecuencia es mucho mayor que en el resto, y especialmente que en la parte final, en donde hay pasos en que la frecuencia no varía. Esta característica es intrínseca del resonador y no puede modificarse a no ser que se cambie la composición interna del elemento resonante para un paso de motor dado.

Si fuera necesario sintonizar este mismo resonador de una manera muy precisa dentro de este primer rango de frecuencias cuya variación es muy grande, sería necesario reajustar el ancho de banda de medida del analizador de redes a un rango menor para aumentar la precisión en la medida, teniendo en cuenta que el mínimo incremento frecuencial viene marcado por el paso del motor, en este caso $7,5^\circ$. Así que también sería necesario cambiar el paso del motor de un paso a medio paso, en cuyo caso el incremento sería de 3.75° .

Llegado este punto debe señalarse que el incremento mínimo entre frecuencias es de 12,5MHz debido a que el cálculo de la frecuencia central se realiza como la frecuencia intermedia de las frecuencias de corte superior e inferior a -3dB (ver apartado 2.2.5). Por lo tanto, como el analizador de redes nos da una precisión de 25MHz con 200 puntos para un rango de 5GHz, si solamente se desplaza una frecuencia de corte (mínimo 25MHz), la frecuencia central o resonante se desplaza 12,5MHz debido a la media realizada en el cálculo, y no 25MHz.

Esta precisión de 25MHz en la medida del analizador de redes es muy correcta, ya que corresponde a un 0,42% de variación respecto a la frecuencia mínima sintonizable (5,9GHz) y únicamente en contadas ocasiones el incremento entre pasos es menor a este valor. Podría caerse en la tentación de decir que el motor elegido tiene un paso excesivo para esta resolución, pero si miramos la parametrización en conjunto, vemos que en su parte final el incremento de frecuencia

entre pasos es muy pequeño y con un motor más preciso se tardaría mucho más en sintonizar estas frecuencias altas.

Por los motivos expuestos anteriormente se entrevé un compromiso entre la resolución del analizador de redes, el paso del motor y la variación frecuencial del resonador escogido. Así que para un óptimo funcionamiento del sistema en futuras aplicaciones, tendría que analizarse cuidadosamente la precisión necesaria en la sintonización del filtro y en función de este parámetro, escoger la resolución del analizador de redes para posteriormente evaluar el mínimo paso necesario del motor en el rango escogido.

En esta parametrización hay algunos casos en que la frecuencia del resonador no varía y como se ha explicado anteriormente, se debe principalmente a que la resolución del analizador de redes es demasiado grande para percibir el incremento de frecuencia debido al giro de un solo paso de $7,5^\circ$. Este fenómeno se observa claramente en la parte final de la parametrización en donde el incremento frecuencial es menor.

También se distinguen en la gráfica dos disminuciones de frecuencia pero si se evalúa en conjunto, este hecho no debe preocuparnos porque se tratan de decrementos insignificantes respecto al aumento acumulado de frecuencia y pueden tener su origen en un movimiento del resonador, el motor o la misma mesa en donde se sitúa toda la estructura.

La tercera y última gráfica de la *Fig. 67* (pérdidas de inserción en cada paso) muestra las pérdidas de inserción o parámetro S_{21} de la matriz S en dB a la frecuencia sintonizada en cada paso. Este parámetro nos da una idea de la cantidad de potencia entrante que se entrega a la salida a la frecuencia sintonizada por el resonador. Cuanto más próximo a cero sea este parámetro mayor potencia entrante será entregada a la salida a la frecuencia sintonizada.

Como puede verse, las pérdidas de inserción se mantiene muy constantes en todo el rango tomando valores entre los -7dB y los -8dB, y solamente en el inicio de la parametrización se consiguen valores próximos a los -10dB. El valor de este parámetro no es un factor limitante a la hora de analizar los resultados de este proyecto, ya que el objetivo es poder diferenciar el desplazamiento de la frecuencia central y, en consecuencia, del pico en la gráfica S_{21} . Aun así, cuanta más potencia se entregue a la salida, más diferencia habrá entre el suelo de ruido y el pico y la medida podrá realizarse con mayor rigurosidad.

5.1.4. Conclusiones de la parametrización

Como conclusión final de esta parametrización, decir que el resonador escogido presta unos resultados excepcionales para medir la frecuencia central y comprobar la variación de la misma al girar el tornillo de sintonización ya que su parámetro S_{21} presenta un pico muy pronunciado a la frecuencia resonante que facilita el análisis.

También cabe mencionar que la resolución escogida es óptima para poder representar todo el rango de frecuencias sintonizables por el resonador sin requerir demasiado tiempo de medida.

Se pone de manifiesto la utilidad de la gráfica generada ya que permite al usuario disponer de la información más relevante del resonador en todo el rango sintonizable y poder localizar las frecuencias óptimas según su aplicación.

Y con los resultados a la vista, se evidencia el excelente comportamiento del software de control, la comunicación con el firmware y el hardware diseñados para este propósito.

5.2. Sintonización descendente de frecuencia

5.2.1. Procedimiento y descripción de la sintonización descendente

La sintonización consiste en ajustar la frecuencia central del resonador a la frecuencia deseada con el menor número de pasos y de la forma más precisa teniendo en cuenta las limitaciones del analizador de redes, el rango de medida, el motor a pasos y la respuesta del resonador.

Como se ha visto en el apartado anterior, la frecuencia a sintonizar debe comprenderse entre los 5,9GHz y los 8,9GHz. Así, la frecuencia elegida es 7GHz ya que se trata de una frecuencia intermedia dentro del rango de sintonización del resonador. El autoajuste se inicia desde una frecuencia aleatoria de 8,325GHz.

Tras introducir los datos y seguir las indicaciones del software de control, se inicia el proceso de sintonizado automático del resonador y una vez terminado se obtienen los resultados expuestos en el siguiente apartado.

5.2.2. Resultados de la sintonización descendente

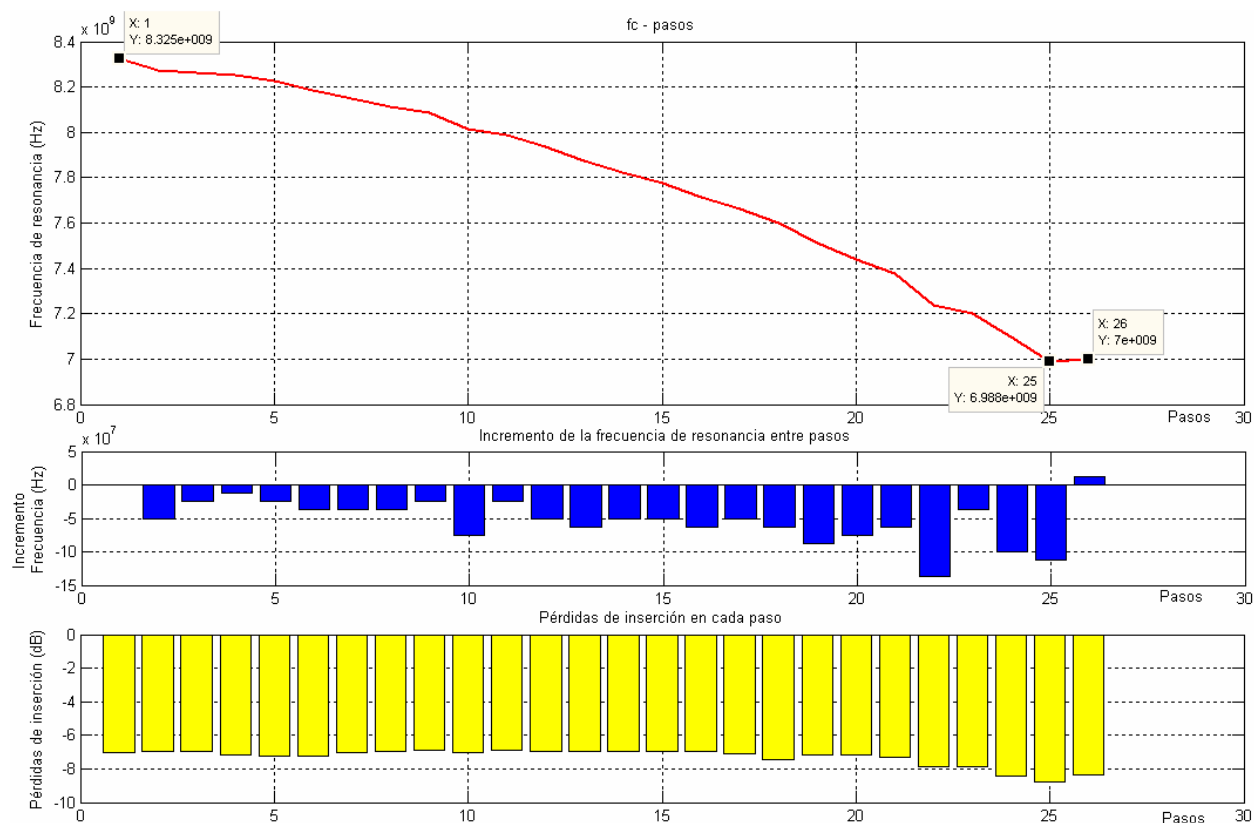


Fig. 74: Resultados de la sintonización descendente del resonador a 7GHz, con un rango de medida del AR de 5GHz a 10GHz y 200 puntos.

En la Fig. 74 se puede ver el resultado de la sintonización del resonador a una frecuencia de 7GHz partiendo desde una frecuencia aleatoria de 8,325GHz. El rango de medida del analizador de redes es de 5GHz a 10GHz con 200 puntos repartidos uniformemente.

Cuando termina la sintonización también se muestran los gráficos de las pérdidas de retorno (S_{11}) y las pérdidas de inserción (S_{21}) a la frecuencia sintonizada final, en este caso 7GHz. Ver Fig. 75.

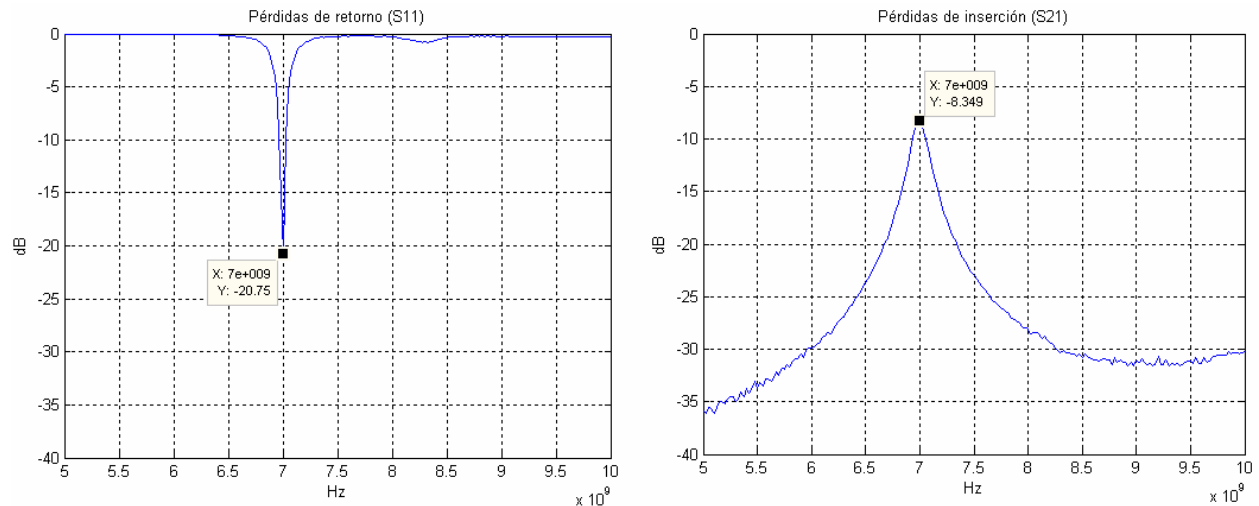


Fig. 75: Pérdidas de retorno (S_{11}) y pérdidas de inserción (S_{21}) del resonador a la frecuencia sintonizada de 7GHz.

5.2.3. Análisis de los resultados

En la primera gráfica de la Fig. 74 se observa claramente el descenso de la frecuencia del resonador hacia la frecuencia deseada de 7GHz. Este hecho confirma el buen comportamiento del software de control que evalúa en todo momento la frecuencia a la que se encuentra el resonador y toma la decisión correcta de bajar la frecuencia. También se confirma el correcto funcionamiento del hardware interpretando correctamente las órdenes del software de control y actuando sobre las fases adecuadas del motor.

Tal y como se ha diseñado el software de control (ver Fig. 53 en pág.73), una vez se rebasa la frecuencia a la que se desea centrar el resonador, que en este caso es en el paso 25, se realiza medio paso en sentido contrario para sintonizar de manera más precisa la frecuencia deseada. Así que en el paso 26 se consigue sintonizar la frecuencia de 7GHz tras medio paso en sentido contrario. Al no existir error en la sintonización, el proceso de autoajuste termina satisfactoriamente.

En la segunda gráfica de la Fig. 74 se observa el decremento de frecuencia entre pasos consecutivos. Esta gráfica da una idea de lo rápido que se acerca el sistema a la frecuencia deseada. Como puede verse, al principio el decremento es menor y aumenta en la parte final. Este resultado es coherente con el resultado obtenido en la parametrización ya que se observó

que a frecuencias inferiores el incremento o decremento entre pasos era mayor. En el último paso el incremento es positivo ya que se realiza medio paso en sentido ascendente de frecuencias para sintonizar más finamente la frecuencia deseada.

En la tercera gráfica de la *Fig. 74* se muestran las pérdidas de inserción a la frecuencia resonante sintonizada en cada paso. Al principio las pérdidas son de -6dB pero aumentan en la parte final hasta los -8dB aproximadamente. Como se ha visto anteriormente, las pérdidas no sufren grandes variaciones y gracias a las modificaciones realizadas en el resonador se mantienen en un nivel superior a los -10dB en todo el rango.

En la *Fig. 75* se observa el gráfico de las pérdidas de retorno (S_{11}) a la frecuencia sintonizada de 7GHz. Como puede verse presenta un pico muy pronunciado a esta frecuencia de -20dB indicando que parte de la potencia entrante penetra hacia el interior de la cavidad resonante y no rebota hacia la fuente como ocurre con el resto de frecuencias presentes a la salida del dispositivo.

Si paralelamente se observa el gráfico de las pérdidas de inserción (S_{21}), se distingue claramente el pico de transmisión a esa misma frecuencia. Este gráfico da información sobre la atenuación que recibe la señal que entra por el puerto 1 y sale por el puerto 2, observándose que la atenuación mínima se da a la frecuencia sintonizada de 7GHz.

Se podrían realizar múltiples estudios de estas gráficas e incluso evaluar sus fases, pero no es el objetivo de este proyecto analizar el comportamiento del resonador, sino comprobar el correcto funcionamiento del sistema diseñado. Aunque gracias a estas graficas se confirma el desplazamiento de la frecuencia central y la sintonización del resonador a la frecuencia deseada.

5.2.4. Conclusiones de la sintonización descendente

A la vista de los resultados obtenidos se puede concluir que se ha conseguido sintonizar correctamente el resonador a la frecuencia deseada de 7GHz de una forma precisa y con el menor número de pasos. Por lo tanto se pone de manifiesto el correcto funcionamiento de todo el sistema, tanto del software de control, el hardware, el firmware y la estructura de soporte.

Resaltar que los incrementos de la frecuencia central del resonador no son iguales en todos los pasos pero este hecho es intrínseco de la respuesta frecuencial del resonador, aunque en este caso se ha dado la circunstancia que en el momento de la sintonización se ha podido lograr la frecuencia exacta.

5.3. Sintonización ascendente de frecuencia

5.3.1. Procedimiento y descripción de la sintonización ascendente

En esta segunda sintonización se desea centrar el resonador a una frecuencia de 8,5GHz partiendo desde una frecuencia aleatoria de 7,488GHz. En este caso la sintonización se realiza ascendentemente para verificar el funcionamiento del sistema tanto en frecuencias inferiores (caso de la sintonización descendente) como superiores.

Como en el caso anterior, la frecuencia a sintonizar debe estar comprendida entre los 5,9GHz y los 8,9GHz ya que es el rango de frecuencias sintonizables por el resonador utilizado.

Tras introducir los datos y seguir las indicaciones del software de control, se inicia el proceso de sintonizado automático del resonador y una vez terminado se obtienen los resultados expuestos en el siguiente apartado.

5.3.2. Resultados de la sintonización ascendente

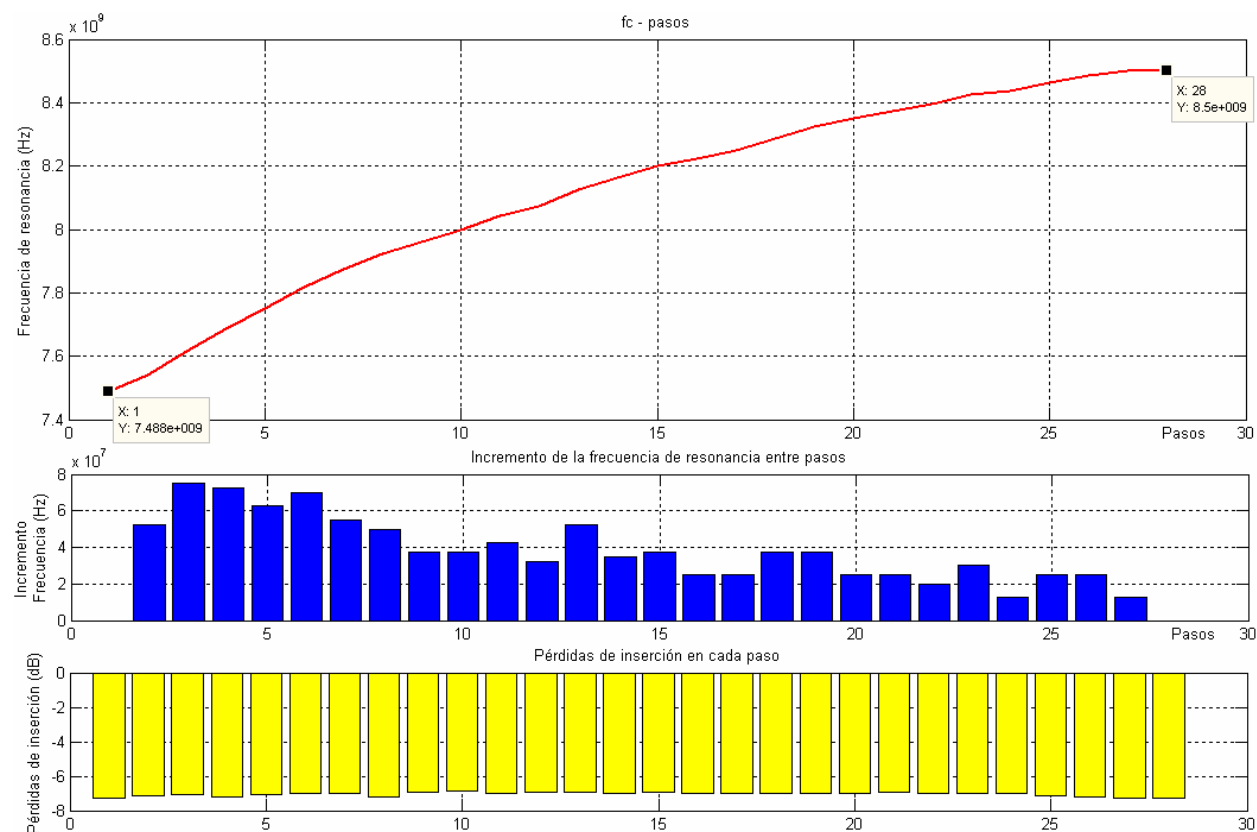


Fig. 76: Resultados de la sintonización ascendente del resonador a 8,5GHz, con un rango de medida del AR de 5GHz a 10GHz y 200 puntos.

En la Fig. 76 se puede ver el resultado de la sintonización del resonador a una frecuencia de 8,5GHz partiendo de una frecuencia aleatoria de 7,488GHz. El rango de medida del analizador de redes es de 5GHz a 10GHz con 200 puntos repartidos uniformemente.

Cuando termina la sintonización también se muestran los gráficos de las pérdidas de retorno (S_{11}) y las pérdidas de inserción (S_{21}) a la frecuencia sintonizada final, en este caso 8,5GHz. Ver Fig. 77.

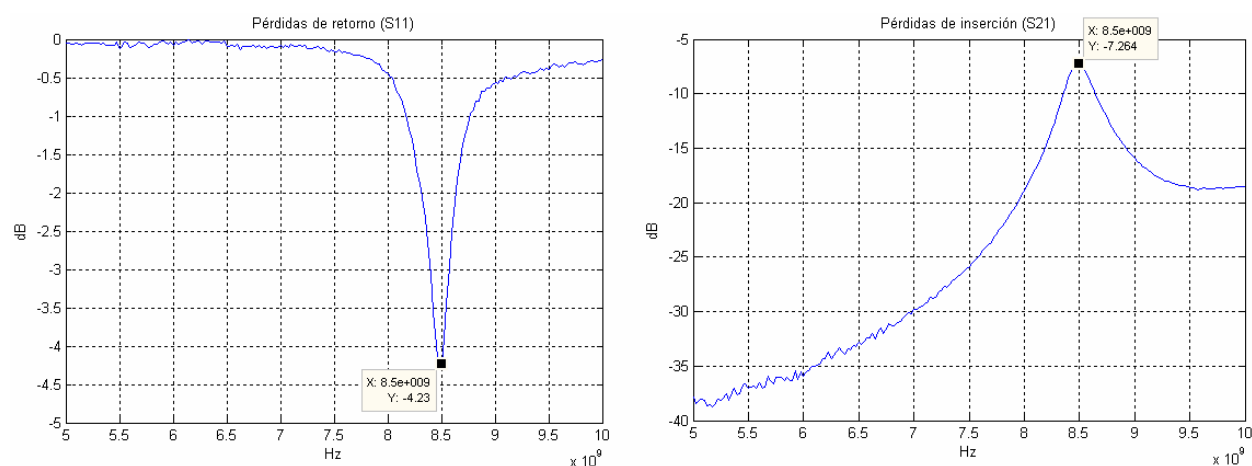


Fig. 77: Pérdidas de retorno (S_{11}) y pérdidas de inserción (S_{21}) del resonador a la frecuencia sintonizada de 8,5GHz.

5.3.3. Análisis de los resultados

En esta segunda sintonización se parte de una frecuencia inferior para terminar ajustando el resonador a una frecuencia de 8,5GHz. Como puede verse en el primer gráfico de la Fig. 76, se distingue claramente la progresión ascendente de la frecuencia hasta llegar a la frecuencia deseada de 8,5GHz.

Al igual que pasaba con la sintonización descendente y la parametrización, el incremento de frecuencia entre pasos consecutivos a frecuencias bajas es mayor, y a medida que se sube en frecuencia el incremento entre pasos disminuye.

Únicamente destacar que en el último medio paso que sirve para sintonizar de una manera más fina la frecuencia deseada, no se observa un decremento de la frecuencia. Este hecho es debido a que, como se ha visto anteriormente, a frecuencias altas el incremento entre pasos es menor, y en este caso el incremento debido a medio paso no es perceptible por el analizador de redes ya que la variación es inferior a la resolución. Aun así, la frecuencia sintonizada es correcta ya que el software de control se acerca al máximo a esta frecuencia antes de empezar el ajuste fino.

En el paso 27 ya se ha conseguido sintonizar la frecuencia deseada y si comparamos este resultado con la gráfica de parametrización (Fig. 67), vemos que el resultado obtenido es

coherente ya que en la parametrización se pasa de 7,5GHz a 8,5GHz con 26 pasos aproximadamente.

Las pérdidas de inserción se mantienen muy constantes entorno a los -7dB y gracias a este hecho se distingue muy claramente el pico de atenuación en el parámetro S_{21} que hace posible el seguimiento de la frecuencia del resonador.

En la Fig. 77 se observa el gráfico de las pérdidas de retorno (S_{11}) y las pérdidas de inserción (S_{21}) a la frecuencia sintonizada de 8,5GHz. Como puede verse ambas presentan un pico muy pronunciado a esta frecuencia, pero en este caso el valor de las pérdidas de retorno es inferior si lo comparamos con la sintonización descendente. Aun así se distingue claramente la frecuencia resonante a la que está sintonizado del resonador.

La gráfica de las pérdidas de inserción presenta un perfil muy similar a la sintonización descendente pero con la excepción de que en este caso el ancho de banda es un poco mayor. Es decir, el pico está más ensanchado en su parte inferior, y de igual forma a -3dB en el ancho de banda. De todas formas el ancho de banda del resonador no es un parámetro importante en estos resultados, ya que el objetivo es el seguimiento de la frecuencia mediante el pico de las pérdidas de inserción.

5.3.4. Conclusiones de la sintonización ascendente

Una vez más, se ha logrado sintonizar el resonador utilizado a la frecuencia deseada de una forma totalmente automática y precisa.

A la vista de los resultados obtenidos, se pone de manifiesto una vez más el correcto funcionamiento de todo el sistema diseñado tanto para la sintonización de frecuencias descendentes como ascendentes.

Se ha verificado la coherencia de los resultados de esta última sintonización con las dos anteriores y se ha visto que en ningún momento la respuesta del resonador se ha visto modificada por el sistema diseñado.

También se ha puesto de manifiesto la estrecha relación entre la respuesta frecuencial del resonador y el paso del motor a la hora de sintonizar finamente las frecuencias, ya que el incremento entre pasos es mayor a frecuencias bajas y menor a frecuencias altas en el resonador utilizado. Y es este mismo hecho el que hace que en el último medio paso de sintonización fina no se incremente suficientemente la frecuencia y no se perciba en el analizador de redes, aunque se haya sintonizado correctamente a la frecuencia deseada gracias al software de control.

6. Conclusiones y líneas de investigación futuras

6.1. Conclusiones generales

Tras las pruebas realizadas tanto al software como al hardware, se llega a la conclusión de que el proyecto ha conseguido satisfacer los objetivos para el cual se ha diseñado. Bien es cierto que llegados a este punto se vislumbran diferentes caminos para seguir mejorando las funcionalidades del mismo, pero el principal objetivo queda logrado: la sintonización automática mediante un motor a pasos de la frecuencia central de un resonador de microondas ajustable por tornillo de sintonización.

El software de control ha respondido adecuadamente tanto en la parametrización como en las sintonizaciones realizadas. En la parametrización se ha recorrido todo el rango de frecuencias sintonizables y se ha establecido el límite inferior y superior del resonador en cuestión. En las sintonizaciones, el sistema ha tomado las decisiones correctas de subir o bajar de frecuencia así como la sintonización más fina posible en los últimos pasos. Y destacar que toda la gestión de errores ha funcionado correctamente tanto por parte del software como del hardware.

El hardware y el firmware han trabajado conjuntamente para actuar adecuadamente sobre las fases del motor a pasos, comunicándose correctamente con el software de control por el puerto serie y gestionando los pulsadores de la placa. Destacar la gran versatilidad que proporciona la programación del microcontrolador por ICSP, ya que no es necesario quitarlo de la placa e insertarlo en otro programador.

La estructura de soporte diseñada ha eliminado completamente el problema de posicionamiento del motor sobre el resonador a sintonizar gracias al ingenioso sistema de suspensión del motor sobre el resonador mediante muelles.

La respuesta del resonador utilizado hace difícil establecer una relación fija entre el paso del motor y la resolución del analizador de redes. Si se desea medir todo el rango de frecuencias sintonizables por el resonador, se precisa de un rango de 5GHz de medida y si se toman 200 puntos para no ralentizar en exceso la medida, la resolución es de 25MHz. Con el motor utilizado de 7,5° se obtienen unos resultados espléndidos para frecuencias altas, aunque el paso resulte un poco excesivo para frecuencias muy bajas.

Para sintonizar filtros con una gran variación de frecuencia, haría falta disminuir el rango de medida al máximo y aumentar, dentro de lo posible, el número de puntos de la medida para obtener una mayor resolución en el analizador de redes. Pero si no se dispone de un motor con un ángulo de paso pequeño, no sirven de nada estas modificaciones.

Para hacer una elección del motor óptimo, primeramente hace falta escoger la resolución del analizador de redes en función de la precisión requerida en la medida. Esta elección viene condicionada por la respuesta frecuencial del filtro ya que depende del rango de frecuencias sintonizables. Por lo tanto únicamente queda a elección del usuario el número de puntos de medida dentro del rango, y este parámetro solamente afecta al tiempo de adquisición de los datos por parte del analizador de redes.

Una vez establecidos el rango de medida y el número de puntos, solamente queda probar el motor elegido y verificar que es capaz de realizar incrementos de frecuencia iguales o inferiores a la precisión requerida en un entorno próximo a la frecuencia deseada. Este hecho se expone con más detalle en el siguiente apartado 6.3.

Solamente apuntar que la utilización del hardware diseñado no es trivial y requiere un estudio previo de los interruptores y *jumpers* para sacarle todo el provecho posible, aunque este mismo hecho hace que disponga de una gran versatilidad a la hora de cambiar de dispositivos, y sobretodo de microcontrolador.

Para terminar las conclusiones generales, comentar que este proyecto ha supuesto un reto ya que partiendo de cero se ha logrado un sistema que interconecta un analizador de redes, un PC, un hardware desarrollado específicamente para este propósito con un microcontrolador, un motor a pasos y un resonador de microondas sintonizable, junto con una estructura de soporte vertical para el motor y la ubicación del hardware.

6.2. Limitaciones del proyecto

La principal limitación del proyecto viene dada por la resolución en la medida. Es indispensable definir una resolución del sistema que sea adecuada al rango de medida del analizador de redes y el ángulo de paso del motor para el dispositivo a sintonizar.

Existen limitaciones físicas en los muelles del soporte del motor. Estos muelles han sido adaptados a la altura del resonador utilizado, pero si se cambia de resonador y con ello su grosor, deben de reajustarse los muelles para que el recorrido del motor sea el óptimo. No obstante, para realizar los experimentos de este trabajo se ha ubicado una espuma debajo del resonador que le da cierta movilidad vertical para que el encaje entre el tornillo de sintonización y la puntera del motor sea perfecto.

Otra cuestión a tener en cuenta aunque no sea una limitación propiamente, es el consumo de los transistores de la etapa de potencia. La demanda de corriente por parte de las fases del motor utilizado es generosa (800mA) y esto causa un calentamiento de los transistores. El límite de corriente de los transistores es de 1A y en algunas ocasiones de manera totalmente esporádica pueden aparecer picos de corriente cercanos a este valor. Es por este motivo que se ha dotado a los transistores de unos disipadores de calor que, a primera vista, pueden parecer sobredimensionados.

6.3. Relación entre el ángulo de paso y la respuesta frecuencial del resonador

La principal limitación de la resolución está causada por el analizador de redes ya que dependiendo del rango de frecuencias a medir y el número de puntos de la medida se puede conseguir mayor o menor resolución. El rango de frecuencias a medir está limitado por la respuesta frecuencial del resonador y el único parámetro ajustable por el usuario que puede afectar a la resolución es el número de puntos de la medida.

En el analizador de redes utilizado para este proyecto se suele elegir normalmente entre dos configuraciones: 200 puntos u 800 puntos. Todas las sintonizaciones y parametrizaciones realizadas se han hecho con 200 puntos ya que el tiempo de obtención de los datos es pequeño. El tiempo de obtención de datos con 800 puntos es muy superior y se consideró que era un tiempo excesivo para la medida.

Con 200 puntos y 5GHz de rango de medida se consigue una resolución de 25MHz. Si el incremento frecuencial entre pasos consecutivos es inferior a la resolución (25MHz) no se aprecia un incremento en la frecuencia sintonizada. Aquí entra en juego el ángulo de paso del motor y la respuesta frecuencial del resonador en el incremento entre frecuencias sintonizables.

Por lo tanto la elección del motor y sobretodo de su ángulo de paso tendría que fundamentarse en la resolución del analizador de redes y en el rango de frecuencias sintonizables por el dispositivo de microondas así como su incremento entre pasos consecutivos. Es decir, no sirve de nada tener una resolución muy grande en el analizador de redes si el paso del motor no está adecuado a la variación frecuencial del dispositivo.

A variaciones grandes de la frecuencial central del dispositivo de microondas utilizado se requiere un paso pequeño en el motor, y a variaciones pequeñas se precisa un paso mayor si no se desea ralentizar el proceso de sintonización y obtener pasos sin variación de frecuencia.

6.4. Líneas de investigación futuras

Llegados a la finalización del proyecto, se entrevén funcionalidades a mejorar que en un futuro pueden realizarse en base a este proyecto. A continuación se comentarán brevemente las más importantes.

La primera y más evidente sería reducir la dependencia de dispositivos externos como el PC. Para prescindir del PC serían necesarios una pantalla de visualización y un teclado numérico para introducir los parámetros y ver la frecuencia central del filtro en cada momento y mostrar información al usuario. Gracias al microcontrolador utilizado y a los puertos de expansión, esta primera mejora no sería muy costosa ya que además, Matlab dispone de traductores de código a C y podría introducirse el software de control del PC dentro del microcontrolador. De esta forma no sería necesaria la comunicación serie entre hardware y PC.

También podría modificarse el software para que no solamente sintonizara la frecuencia central del dispositivo, sino que también pudiera ajustar el ancho de banda, la posición de los ceros de transmisión, la atenuación, etc.

Siguiendo con la anterior mejora, podría implementarse un banco de motores a pasos capaces de ajustar varios parámetros a la vez. De esta forma podría configurarse el dispositivo de microondas sin necesidad de desacoplar el motor del tornillo de sintonización de cada parámetro. O incluso se podría robotizar el soporte del motor para que se moviera por encima del dispositivo y fuera ajustando de manera automática cada parámetro. Esta versión más futurista conllevaría el desarrollo de un software enorme para el posicionamiento del motor en tres dimensiones, un elevado tiempo de sintonía y un problema difícil de solucionar: el alineamiento de la ranura del tornillo con la punta del motor.

Como puede verse, hay muchas vías de investigación futuras partiendo del proyecto actual, pero como siempre o casi siempre, dependen de las mismas variables: tiempo y dinero.

Bibliografía y recursos web

Bibliografía

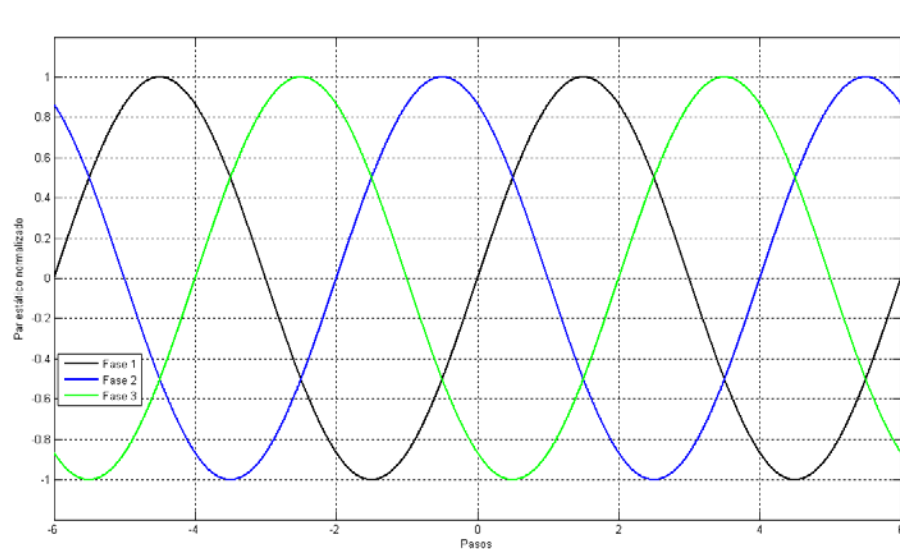
- [1] TORRENTS Y DOLZ, Josep Maria; SORIANO FOSAS, David. *Estudi de viabilitat, disseny i construcció d'un sistema d'avanç de paper per un traçador gràfic accionat per motor pas a pas*. Proyecto fin de carrera. Universitat Politècnica de Catalunya, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona. Barcelona, 1996.
- [2] PALACIOS MUNICIO, Enrique; REMIRO DOMINGUEZ, Fernando; LOPEZ PEREZ, Lucas Jose. *Microcontrolador PIC16F84: Desarrollo de proyectos*. Ra-Ma Editorial. 2ª edición. Madrid, 2005.
ISBN 978-84-7897-691-1
- [3] GARCÍA LAMPÉREZ, Alejandro. *Métodos avanzados de síntesis y optimización de filtros y multiplexores de microondas*. Tesis Doctoral. Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación de Madrid. Madrid, 2004.
- [4] VALDÉS PÉREZ, Fernando E.; PALLÀS ARENY, Ramón. *Microcontroladores: Fundamentos y aplicaciones con PIC*. Editorial Marcombo. 1ª edición. Barcelona, 2007.
ISBN 84-2671-414-5
- [5] TIPLER, Paul Allen; MOSCA, Gene. *Física: para la ciencia y la tecnología*. Vol. 2. Editorial Reverté. 6ª edición. Barcelona, 2010.
ISBN 978-84-2914-430-7
- [6] PRAT, Lluís; et al. *Laboratorio de Electrónica. Curso básico*. Aula Politècnica 29. Edicions UPC. 4ª edición. Barcelona, 2001.
ISBN 84-8310-425-4
- [7] PEÑA BASURTO, Marco A.; CELA ESPÍN, José M. *Introducción a la programación en C*. Aula Politècnica. Edicions UPC. 1ª edición. Barcelona, 2000.
ISBN 84-8301-429-7
- [8] GUASCA, A.; et al. *Laboratori de Comunicacions 3 (LC3): Transparències*. CPET Ref. 16811. Universitat Politècnica de Catalunya, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament de Teoria del Senyal i Comunicacions. Barcelona.

- [9] COMERÓN, A.; et al. *Transparencias de la asignatura de Microondas*. Universitat Politècnica de Catalunya, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament de Teoria del Senyal i Comunicacions. Barcelona.
- [10] SAX, H. *Stepper Motor Driving*. Designer's guide to power products. Application manual. SGS-Thomson Microelectronics. 2nd edition. 1992.
- [11] CONDIR, Reston. *Stepper Motor Control Using the PIC16F684*. AN906. Microchip Technology Inc., 2004.
- [12] YEDAMALE, Padmaraja; CHATTOPADHYAY, Sandip; *Stepper Motor Microstepping with PIC18C452*. AN288. Microchip Technology Inc., 2002.
- [13] "Faster Switching" for "Standard Couplers". Document Number: 83590. Vishay Intertechnology, Inc., 2008.
- [14] *Application Examples*. Document Number: 83741. Vishay Intertechnology, Inc., 2008.
- [15] CONDIR, Reston; Dr. JONES, Douglas W.. *Stepping Motors Fundamentals*. AN907. University of Iowa. Microchip Technology Inc., 2004.
- [16] *Stepper-Motor Performance: Constant-Current Chopper Drive UPS*. AN486. STMicroelectronics, 2003.
- [17] *Motor Control: Reference Guide*. Order Code: BRMOTOR0509. STMicroelectronics, Italy, 2003.
- [18] *DR-38: Step Motor Driver Users Guide*. Advanced Micro Systems, Inc., 2003.
- [19] *Agilent: Network Analyzer Basics*. Publication Number: 5965-7917E. Agilent Technologies, Inc. USA, 2004.
- [20] *Agilent 8510XF Vector Network Analyzer Single-Connection, Single-Sweep Systems*. Publication Number: 5965-9888E. Agilent Technologies, Inc. USA, 2006.
- [21] *Agilent AN 1287-1: Understanding the Fundamental Principles of Vector Network Analysis*. Publication Number: 5965-7707E. Agilent Technologies, Inc. USA, 2000.
- [22] *Agilent: Specifying Calibration Standards for Agilent 8510 Network Analyzer*. AN 8510-5B. Publication Number: 5956-4352E. Agilent Technologies, Inc. USA, 2006.
- [23] *Agilent: 10 Hints for Making Better Network Analyzer Measurements*. AN 1291-1B. Publication Number: 5965-8166E. Agilent Technologies, Inc. USA, 2001.
- [24] *Agilent AN 1287-2: Exploring the Architectures of Network Analyzers*. Publication Number: 5965-7708E. Agilent Technologies, Inc. USA, 2000.

Páginas web consultadas

- [25] <http://www.cs.uiowa.edu>
- [26] <http://www.me.umn.edu>
- [27] <http://www.stepcontrol.com>
- [28] <http://www.microchip.com>
- [29] <http://zone.ni.com>
- [30] <http://www.st.com>
- [31] <http://www.vishay.com>
- [32] <http://www.ti.com>
- [33] <http://www.maxim-ic.com>
- [34] <http://www.national.com>
- [35] <http://www.renesas.eu>
- [36] <http://electrosofts.com>
- [37] <http://www.home.agilent.com>
- [38] <http://www.sxlist.com>
- [39] <http://redeya.com/electronica/>
- [40] <http://ams2000.com>
- [41] <http://www.todorobot.com.ar>
- [42] <http://gearmotorblog.files.wordpress.com>

Anexo A.: Ecuación de la gráfica 'Par estático normalizado'



Página 18, Fig. 11: Gráfico del par estático normalizado de un motor a pasos con 3 fases y la fase 1 como referencia o paso cero.

Para realizar la gráfica del par estático normalizado, se ha partido de una gráfica extraída de la referencia bibliográfica [1] página 28, y se ha singularizado para el caso de tener un motor de 7.5° de paso. Partiendo de la ecuación 2.3, se extrapola la ecuación 2.6 y se particularizó para el caso de este proyecto con el motor elegido.

Posteriormente y con la ayuda del Matlab, se dibujó la gráfica en cuestión. El código introducido en Matlab para obtener la gráfica es el siguiente:

```
a=-45:0.01:45;

plot(a/7.5, sin((2*pi*a)/(2*3*7.5)), 'k', a/7.5, sin(((2*pi*a)/(2*3*7.5))+(2*pi*1)/3), 'b',
a/7.5, sin(((2*pi*a)/(2*3*7.5))+(2*pi*2)/3), 'g', 'LineWidth', 2);

grid on; xlabel('Pasos'); ylabel('Par estático normalizado'); axis([-6 6 -1.2 1.2]);

legend('Fase 1', 'Fase 2', 'Fase 3');
```

La variable a actúa como ángulo para dibujar los senos y se le asigna un rango de 90° con el cero en la mitad para que se representen pasos anteriores y futuros al paso actual. Posteriormente se dibuja la gráfica de las tres fases siguiendo la ecuación 2.6, asignando diferentes colores a cada una ('k' para el negro, 'b' para el azul y 'g' para el verde). Seguidamente se etiquetan y limitan los ejes y se añade la leyenda del gráfico.

Anexo B.: Código del Software de Control

En las siguientes páginas se encuentran los códigos de los archivos componentes del software programados en entorno y lenguaje Matlab.

El orden de los archivos es el siguiente:

- *main_sintonizacion.m*
- *inicializar_puerto.m*
- *petición_datos.m*
- *inicializar_filtro.m*
- *calcular_S.m*
- *enviar.cpp*
- *rebre.cpp*
- *calcular_fc.m*
- *calcular_fc_simulacion.m*
- *envia_orden.m*
- *finalizar_puerto.m*

Archivo *main_sintonizacion.m*

```
% main_sintonizacion
% Esta función se encarga de realizar todo el control de la sintonización y
% la recuperación y control de errores. Es el hilo principal del programa y
% todas las funciones vuelven a ella, tanto si hay errores como si no.

function main_sintonizacion

clc;
clear;

% Declaración de las variables globales ya que diferentes archivos *.m tendrán
% que acceder a ellas y/o modificar su valor de manera NO concurrente.

global directorio % Directorio actual.
global directorio_filtros % Directorio en donde guardar las variables y parámetros.
global archivo_filtro % Nombre del archivo con los parámetros del filtro.

global fmin % Frecuencia mínima en la medida del Analizador.
global fmin % Frecuencia mínima sintonizable por el filtro.
global fmax % Frecuencia máxima en la medida del Analizador.
global fmax % Frecuencia máxima sintonizable por el filtro.
global fcentrar % Frecuencia a centrar el filtro.

global Npuntos % Número de puntos de medida del Analizador.
global adress % Dirección GPIB del Analizador de redes.

global freq % Vector de frecuencia entre fmin y fmax.
global S % Matriz con freq y los parámetros S21 y S11.
global fc % Frecuencia central del filtro.
global s21max % Valor de las pérdidas de inserción a fc.
global WB % Ancho de banda en la banda de paso.
global s11 % Vector con los parámetros S11.
global s21 % Vector con los parámetros S21.

global serie % Puerto de comunicaciones RS232.
global input_RS232 % Byte leído en puerto RS232.

global Modo % Paso completo (full) o medio paso (half).
global Direccion % Direccion de giro del motor (derecha o izquierda).

global indice; % Índice para recorrer los vectores de frecuencias.
indice=1;
%-----Simulación-----
% global fcv % Vector de fc's para la simulación.
% global puntero_fcv % Puntero del vector fcv.
%-----

%-----Requisitos previos-----
fprintf('Este programa se encarga de sintonizar automáticamente cualquier filtro de microondas.\n');
fprintf('\nPara conectar correctamente el hardware, siga los pasos que a continuación se describen.\n');
fprintf('\t1.-Primero de todo, compruebe que la configuración del hardware sea la correcta (interruptores y
jumpers).\n');
fprintf('\t2.-A continuación conecte la alimentación del hardware a +12V (terminal rojo) y 0V (terminal
negro).\n');
```

```

fprintf('t También puede conectar el conector de 12V tipo DC (Recomendado).\n');
fprintf('t3.-Conecte el cable serie al conector DB-9 del hardware y al puerto COM1 de este PC.\n');
fprintf('t4.-Encienda el hardware mediante el interruptor que se encuentra al lado del conector rojo.\n');
fprintf('tSe debe encender el led verde que se encuentra próximo al conector.\n');
fprintf('nPara situar correctamente el filtro en el hardware, siga los pasos que se describen a continuación:\n');
fprintf('t5.-Desconecte los cables verdes del motor.\n');
fprintf('t6.-Saque, tirando hacia arriba, la plancha del hardware que contiene el motor. Tenga cuidado con las 4
arandelas.\n');
fprintf('t7.-Sitúe el filtro en la ubicación de medida y conecte los cables coaxiales del analizador.\n');
fprintf('t8.-Vuelva a insertar la plancha que contiene el motor encajando el tornillo del filtro en la puntera del
motor.\n');
fprintf('nSi ha seguido correctamente todos los pasos, ya puede iniciar el proceso de sintonización automática del
filtro.\n');
fprintf('Pulse cualquier tecla para continuar.\n');
pause;
% -----Primera fase-----
try
% Se inicializa la comunicación con el motor.
inicializar_puerto;

% Obtención de los datos.
peticion_datos;

% Inicialización del filtro si es necesario.
fprintf('nEscoja un modo de sintonización (default: 3):\n');
fprintf('t1-Sintonización automática con parametrización previa del filtro.\n');
fprintf('t2-Sintonización automática con carga de los parámetros del filtro desde un archivo.\n');
fprintf('t3-Sintonización automática sin parámetros del filtro.\n>>');
reply=input(', 's');
if ((isempty(reply))||((reply~='1')&&(reply~='2')&&(reply~='3'))))
    reply=3;
end
switch reply
case '1'
    inicializar_filtro;
case '2'
    fprintf('nIntroduzca la ruta completa del archivo a cargar con los parámetros del filtro.\n (Default: %s)\n>>',
archivo_filtro);
    dir_carga=input(', 's');
    if (isempty(dir_carga))
        dir_carga=archivo_filtro;
    end
    while(exist(dir_carga)==0)
        fprintf('El archivo a cargar no existe. Por favor, introduzca un archivo válido o pulse "N" para
parametrizar el filtro: ');
        dir_carga=input(', 's');
        if(lower(dir_carga)=='n')
            inicializar_filtro;
            dir_carga=archivo_filtro;
        end
    end
    load(dir_carga);
case '3'
    fmin=fmin;
    fmax=fmax;
otherwise
end

%-----Simulación-----
% Se crea un vector de fc's que simula las frecuencias centrales

```

```

% por las cuales nos iremos desplazando cuando movamos el motor. Es
% como si se desplazara la fc real del filtro.
% fcv=linspace(fmin,fmax,300);

% Definimos el puntero del vector fcv. Se empieza por la mitad del
% vector a no ser que se introduzca un valor en concreto.
% puntero_fcv=(length(fcv)/2);

% -----Segunda fase-----
% Control y monitorización del motor hasta el ajuste total.

% Primero se calcula el parámetro S21 del filtro para posteriormente
% calcular la frecuencia central del mismo.
calcular_S;
calcular_fc;
% fc=fcv(puntero_fcv); % sustituye a las dos anteriores en simulación.

%-----Gráficas-----
v_fc(indice)=fc;
v_s21max(indice)=s21max;
if (indice==1)
    v_dif(indice)=0;    % No hay frecuencia anterior a comparar.
else
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
end
v_WB(indice)=WB;
indice=indice+1;
%-----

% Se empieza el Auto-ajuste con pasos enteros (Modo FULL) y dependiendo
% de la frecuencia a sintonizar, se sube o baja la frecuencia.
% Posteriormente se pasa a Modo HALF para afinar más la sintonización y
% nos quedamos en el medio paso superior o inferior que nos proporcione
% el menor error absoluto.

Modo='full'
if (fc > fcentrar)
    Direccion='derecha'
    while (fc > fcentrar)
        fup=fc;
        envia_orden;
        calcular_S;
        calcular_fc;
        %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
        fdw=fc;
        if(fc<fmin)    %Hemos llegado al límite inferior del filtro
            error('MyToolbox:LIMITE_INFERIOR', 'Se ha llegado al límite físico inferior del filtro establecido por el
parámetro fmin.\n');
        end
        %-----Gráficas-----
        v_fc(indice)=fc;
        v_s21max(indice)=s21max;
        v_dif(indice)=v_fc(indice)-v_fc(indice-1);
        v_WB(indice)=WB;
        indice=indice+1;
        %-----
    end

    Modo='half'
    Direccion='izquierda'

```

```

envia_orden;
calcular_S;
calcular_fc;
%calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
fmid=fc;
%-----Gráficas-----
v_fc(indice)=fc;
v_s2lmax(indice)=s2lmax;
v_dif(indice)=v_fc(indice)-v_fc(indice-1);
v_WB(indice)=WB;
indice=indice+1;
%-----

if(abs(fup-fcentrar) < abs(fcentrar-fmid))
    Direccion='izquierda'
    envia_orden;
    calcular_S;
    calcular_fc;
    %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
    %-----Gráficas-----
    v_fc(indice)=fc;
    v_s2lmax(indice)=s2lmax;
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
    v_WB(indice)=WB;
    indice=indice+1;
    %-----

elseif(abs(fcentrar-fdw) < abs(fcentrar-fmid))
    Direccion='derecha'
    envia_orden;
    calcular_S;
    calcular_fc;
    %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
    %-----Gráficas-----
    v_fc(indice)=fc;
    v_s2lmax(indice)=s2lmax;
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
    v_WB(indice)=WB;
    indice=indice+1;
    %-----

else
    % Si el error es más pequeño en el medio paso actual, no
    % hacemos nada.
end

elseif (fc < fcentrar)
    Direccion='izquierda'
    while (fc < fcentrar)
        fdw=fc;
        envia_orden;
        calcular_S;
        calcular_fc;
        %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
        fup=fc;
        if(fc>ffmax) %Hemos llegado al límite superior del filtro
            error('MyToolbox:LIMITE_SUPERIOR', 'Se ha llegado al límite físico superior del filtro establecido por
el parámetro fmax.\n');
        end
        %-----Gráficas-----

```

```

    v_fc(indice)=fc;
    v_s21max(indice)=s21max;
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
    v_WB(indice)=WB;
    indice=indice+1;
    %-----
end

Modo='half'
Direccion='derecha'
envia_orden;
calcular_S;
calcular_fc;
%calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
fmid=fc;
%-----Gráficas-----
    v_fc(indice)=fc;
    v_s21max(indice)=s21max;
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
    v_WB(indice)=WB;
    indice=indice+1;
    %-----

if(abs(fcentrar-fdw) < abs(fmid-fcentrar))
    Direccion='derecha'
    envia_orden;
    calcular_S;
    calcular_fc;
    %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
    %-----Gráficas-----
        v_fc(indice)=fc;
        v_s21max(indice)=s21max;
        v_dif(indice)=v_fc(indice)-v_fc(indice-1);
        v_WB(indice)=WB;
        indice=indice+1;
        %-----

elseif(abs(fup-fcentrar) < abs(fmid-fcentrar))
    Direccion='izquierda'
    envia_orden;
    calcular_S;
    calcular_fc;
    %calcular_fc_simulacion; % sustituye a las dos anteriores en simulación.
    %-----Gráficas-----
        v_fc(indice)=fc;
        v_s21max(indice)=s21max;
        v_dif(indice)=v_fc(indice)-v_fc(indice-1);
        v_WB(indice)=WB;
        indice=indice+1;
        %-----

else
    % Si el error es más pequeño en el medio paso actual, no
    % hacemos nada.
end

else
    fprintf('La frecuencia deseada es exactamente la sintonizada en el filtro.\n');
end

```



```

fprintf('\nEl filtro se ha sintonizado correctamente a la frecuencia de: %d Hz.\n', fc);

%Cerramos la comunicación con el PIC
finalizar_puerto;

%-----Tercera fase-----
%Presentación de los resultados en pantalla
%Obtenemos la medida de la pantalla y creamos una ventana para situar
%las gráficas.
scrz = get(0,'ScreenSize');
figure('Position',[scrz(3)/5 scrz(4)/7 4*scrz(3)/5 6*scrz(4)/7]);
subplot(4,1,1); plot(v_fc,'r','LineWidth',2); grid on; xlabel('Pasos'); ylabel('Frecuencia Central (Hz)'); title('fc -
pasos');
subplot(4,1,2); bar(v_dif,'b'); grid on; xlabel('Pasos'); ylabel('Frecuencia (Hz)'); title('Estudio del incremento de
frecuencia entre pasos');
subplot(4,1,3); bar(v_WB,'g'); grid on; xlabel('Pasos'); ylabel('Ancho de banda (Hz)'); title('Ancho de banda en
cada paso');
subplot(4,1,4); bar(v_s21max,'y'); grid on; xlabel('Pasos'); ylabel('Pérdidas de inserción (dB)'); title('Pérdidas de
inserción en cada paso');

% Se mide el parámetro S11 para las gráficas.
enviar(address,'S11;SING;FORM2;OUTPDATA');
[s11]=rebre(address);
S(:,3)=20*log10(abs(s11));

figure('Position',[scrz(3)/5 scrz(4)/7 4*scrz(3)/5 6*scrz(4)/7]);
subplot(2,2,1); plot(freq,S(:,3)); grid; ylabel('dB'); xlabel('Hz'); title('Pérdidas de retorno (S11)');
subplot(2,2,3); plot(freq,(180/pi)*phase(s11)); grid; ylabel('Phase S11 (deg)'); xlabel('Hz'); title('Fase en la
entrada');
subplot(2,2,2); plot(freq,S(:,2)); grid; ylabel('dB'); xlabel('Hz'); title('Pérdidas de inserción (S21)');
subplot(2,2,4); plot(freq,(180/pi)*phase(s21)); grid; ylabel('Phase S21 (deg)'); xlabel('Hz'); title('Fase en la
salida');

%Guardamos el valor final de las variables para su posible revisión.
archivo_sintonizacion=[directorio_filtros '\' 'ultima_sintonizacion' '.mat'];
save(archivo_sintonizacion);
save(archivo_filtro, 'ffmin', 'ffmax');

%-----Control de errores-----
catch
[errmsg, msg_id] = lasterr;
switch (msg_id)
case 'MyToolbox:PIC_NO_OK'
fprintf('\nNo se puede realizar la comunicación porque el PIC no responde.\nCompruebe la alimentación del
hardware y reinicie la aplicación.\n');

case 'MyToolbox:PIC_NO_TX_OK'
fprintf('\nNo se puede realizar la comunicación porque el PIC no responde correctamente.\nCompruebe la
alimentación del hardware y reinicie la aplicación.\n');

case 'MATLAB:serial:fopen:opfailed'
fprintf('\nEl puerto serie COM1 no está disponible.\nEjecute finalizar_puerto en la entrada de comandos para
cerrar el puerto serie COM1 o reinicie toda la aplicación.\n');

case 'MyToolbox:NO_f3dbup'
fprintf('\nNo se ha encontrado frecuencia de corte superior a 3dB en el margen especificado.\n');
fprintf('Esto puede deberse a:\n\tLa frecuencia superior Fmax es demasiado pequeña.\n\tEl filtro es
demasiado plano en el rango de frecuencias [Fmin-Fmax] y no cae 3dB.\n');
fprintf('Reinicie la aplicación con una frecuencia superior Fmax mayor.\n');

```

```

case 'MyToolbox:NO_f3dbdw'
    fprintf('\nNo se ha encontrado frecuencia de corte inferior a 3dB en el margen especificado.\n');
    fprintf('Esto puede deberse a:\n\tLa frecuencia inferior Fmin es demasiado grande.\n\tEl filtro es demasiado
plano en el rango de frecuencias [Fmin-Fmax] y no cae 3dB.\n');
    fprintf('Reinicie la aplicación con una frecuencia inferior Fmin menor.\n');

case 'MyToolbox:NO_FULL_LEFT'
    fprintf('\nNo se ha podido sintonizar la frecuencia deseada porque se ha pulsado el paro de
EMERGENCIA.\nPor favor vuelva a intentarlo.\n');
    fprintf('La última frecuencia sintonizada ha sido: %d Hz.\n', fc);

case 'MyToolbox:NO_FULL_RIGHT'
    fprintf('\nNo se ha podido sintonizar la frecuencia deseada porque se ha pulsado el paro de
EMERGENCIA.\nPor favor vuelva a intentarlo.\n');
    fprintf('La última frecuencia sintonizada ha sido: %d Hz.\n', fc);

case 'MyToolbox:NO_HALF_LEFT'
    fprintf('\nNo se ha podido sintonizar la frecuencia deseada porque se ha pulsado el paro de
EMERGENCIA.\nPor favor vuelva a intentarlo.\n');
    fprintf('La última frecuencia sintonizada ha sido: %d Hz.\n', fc);

case 'MyToolbox:NO_HALF_RIGHT'
    fprintf('\nNo se ha podido sintonizar la frecuencia deseada porque se ha pulsado el paro de
EMERGENCIA.\nPor favor vuelva a intentarlo.\n');
    fprintf('La última frecuencia sintonizada ha sido: %d Hz.\n', fc);

case 'MyToolbox:INVALID_DATA'
    fprintf('\nEl PIC ha devuelto un dato NO válido.\n');
    fprintf('Cancele la comunicación con el PIC, resetee el hardware, compruebe que las frecuencias se
encuentran en el rango y vuelva a reiniciar la aplicación.\n');

case 'MyToolbox:MODO_ERROR'
    fprintf('\nLa variable MODO tiene un valor incorrecto.\nPor favor, resetee el hardware y reinicie todo el
proceso de nuevo.\n');

case 'MyToolbox:DIRECCION_ERROR'
    fprintf('\nLa variable DIRECCION tiene un valor incorrecto.\nPor favor, resetee el hardware y reinicie todo
el proceso de nuevo.\n');

case 'MyToolbox:PIC_NO_KO'
    fprintf('\nNo se puede cerrar la comunicación con el hardware porque el PIC no responde.\nPor favor,
resetee el hardware y reinicie todo el proceso de nuevo.\n');

case 'MyToolbox:PIC_NO_TX_KO'
    fprintf('\nNo se puede cerrar la comunicación con el hardware porque el PIC no responde
correctamente.\nPor favor, resetee el hardware y reinicie todo el proceso de nuevo.\n');

case 'MyToolbox:LIMITE_INFERIOR'
    fprintf('\nSe ha alcanzado el límite físico inferior del filtro.\n');
    fprintf('La última frecuencia sintonizada ha sido %d Hz.\n', fc);
    fprintf('Asegúrese de que el filtro no haya alcanzado su frecuencia mínima o\nque el archivo de carga de los
parámetros del filtro sea el correcto.\n');
    fprintf('Si el filtro no ha alcanzado su frecuencia inferior, compruebe que el tornillo de sintonización no esté
atascado.\n');
    fprintf('Reinicie todo el proceso.\n');

case 'MyToolbox:LIMITE_SUPERIOR'
    fprintf('\nSe ha alcanzado el límite físico superior del filtro.\n');
    fprintf('La última frecuencia sintonizada ha sido %d Hz.\n', fc);

```

```
fprintf('Asegúrese de que el filtro no haya alcanzado su frecuencia máxima o\nque el archivo de carga de los
parámetros del filtro sea el correcto.\n');
fprintf('Si el filtro no ha alcanzado su frecuencia superior, compruebe que el tornillo de sintonización no esté
atascado.\n');
fprintf('Reinicie todo el proceso.\n');

case 'MyToolbox:ffmin<Fmin'
    fprintf('No se puede iniciar la parametrización del filtro.\nLa frecuencia de inicio de medida (Fmin) del
Analizador es mayor que la frecuencia mínima sintonizada por el filtro.\n');
    fprintf('Cambie la frecuencia de inicio de medida del Analizador (Fmin) o suba un poco el tornillo de
sintonización.\n');
    fprintf('Posteriormente reinicie todo el proceso.');
```

```
otherwise
    fprintf('\nSe ha producido el siguiente error:\n');
    msg_id
    errmsg
    fprintf('Resetea el hardware y reinicie la aplicación.\n');
end

%Cerramos la comunicación con el PIC
finalizar_puerto;

end
```

Archivo *inicializar_puerto.m*

```
%inicializar_puerto
% Función que crea un objeto tipo puerto serie en matlab y se conecta al
% puerto serie del PC 'COM1'. Posteriormente se asegura de que la conexión
% con el motor se ha efectuado correctamente y espera a que el motor esté
% preparado.

function inicializar_puerto

global serie
global input_RS232

% Se crea el objeto puerto serie
serie=serial('COM1');

% Configuración del puerto serie como el del PIC
set(serie,'BaudRate',9600);
set(serie,'DataBits',8);
set(serie,'Parity','none');
set(serie,'StopBits',1);
set(serie,'FlowControl','none');

% Se abre el puerto para iniciar la comunicación.
fopen(serie);

% Informa al PIC que la comunicación ya está lista.
fprintf('Conectando con el motor...\n');
fprintf(serie, '%c', 128);

% Espera la respuesta del PIC. Si está listo empieza la conexión. Si
% no, ocurre un error y sale del programa.
input_RS232=fread(serie, 1)

if input_RS232== 128
    fprintf('Conexión realizada con éxito.\n');
elseif isempty(input_RS232)
    error('MyToolbox:PIC_NO_OK', 'No se puede realizar la comunicación: El PIC no responde.');
```

```
else
    error('MyToolbox:PIC_NO_TX_OK', 'No se puede realizar la comunicación: El PIC no responde
correctamente.');
```

```
end

return
```

Archivo *peticion_datos.m*

```
%peticion_datos
% Función que pide al usuario los datos necesarios para realizar la
% medida del filtro en el Analizador de redes y el directorio para guardar
% las variables y los datos obtenidos en la medida.
%
% Debe de introducirse:
% directorio en donde guardar los datos
% nombre del archivo a guardar (*.mat)
% frecuencia mínima
% frecuencia máxima
% frecuencia a centrar el filtro
% número de puntos de la medida
% dirección GPIB del Analizador de redes

function peticion_datos

global directorio
global directorio_filtros
global archivo_filtro

global fmin
global fmax
global fcentrar
global Npuntos
global adress
global freq

directorio = pwd;

% Petición al usuario del directorio para guardar las variables.
fprintf("\nEl directorio en donde se guardarán los datos del filtro es: %s\n", directorio);
fprintf("Tenga en cuenta que dentro de esta carpeta se creará una nueva carpeta llamada\n"sintonizacion_filtros".\n");
reply=input("Desea cambiar el directorio en donde se guardarán los datos del filtro [Y/N]? ' ', 's');
while ((isempty(reply)) || ((lower(reply)~= 'y') && (lower(reply)~= 'n'))))
    reply=input("Debe introducir Y o N.\nDesea cambiar el directorio actual? ', 's');
end

switch lower(reply)
    case 'y'
        fprintf('Introduzca el directorio para guardar los datos del filtro (default: %s):', directorio);
        reply=input(", 's');
        if (isempty(reply))

            elseif (exist(reply, 'dir')==0)
                mkdir(reply);
                directorio=reply;
            else
                directorio=reply;
            end
        case 'n'
        otherwise
        end
end
```

```

% Se crea, si es necesario, el directorio para guardar los datos de los filtros.
directorio_filtros=[directorio 'sintonizacion_filtros'];
if (exist(directorio_filtros, 'dir')==0)
    mkdir(directorio, 'sintonizacion_filtros');
end

% Se crea el archivo que guardará las variables del filtro.
nombre_filtro=input('Introduzca el nombre del filtro (default: filtro_casi_eliptico):','s');
if ((isempty(nombre_filtro)))
    nombre_filtro='filtro_casi_eliptico';
end
archivo_filtro=[directorio_filtros '\ nombre_filtro '.mat'];

% Petición al usuario de la frecuencia mínima y frecuencia máxima para configurar el Analizador de redes.
finf=input('Introduzca la frecuencia inferior Fmin del analizador de redes(GHz): ');
while isempty(finf)
    finf=input('Introduzca una frecuencia inferior válida(GHz): ');
end
fmin=finf*1e9;      % Conversión a GHz

fsup=input('Introduzca la frecuencia superior Fmax del analizador de redes(GHz): ');
while (isempty(fsup)) || (fsup <= finf)
    fsup=input('Introduzca una frecuencia superior válida(GHz): ');
end
fmax=fsup*1e9;      % Conversión a GHz

% Petición al usuario de la frecuencia central deseada.
fdeseada=input('Introduzca la frecuencia a la que desea centrar el filtro(GHz): ');
while (isempty(fdeseada)) || (fdeseada < finf) || (fdeseada > fsup)
    fdeseada=input('Introduzca una frecuencia comprendida entre Fmin y Fmax (GHz): ');
end
fcentrar=fdeseada*1e9; % Conversión a GHz

% Petición al usuario del número de puntos para configurar el Analizador de redes.
Npuntos=input('Introduzca el número de puntos del analizador(máximo: 801, default: 201): ');
if isempty(Npuntos)
    Npuntos = 201;
end

% Petición al usuario de la dirección GPIB del Analizador de redes.
adress=input('Introduzca la dirección GPIB del analizador (default: 16): ');
if isempty(adress)
    adress = 16;
end

% Se crea el vector frecuencia entre fmin y fmax con Npuntos entre las dos
freq=linspace(fmin,fmax,Npuntos);

% Se guardan las variables
save(archivo_filtro);

return

```

Archivo *inicializar_filtro.m*

```
%inicializar_filtro
% Esta función se encarga de medir los parámetros del filtro haciendo un
% barrido completo dentro del rango de frecuencias del Analizador de redes
% [Fmin-Fmax]. Sirve principalmente para encontrar las frecuencias mínima y
% máxima sintonizables por el filtro.

function inicializar_filtro

global archivo_filtro

global fmin
global fmin
global fmax
global fmax
global fc
global WB
global s21max

global Modo
global Direccion

global indice      % Índice para recorrer los vectores de frecuencias.
repeticion=6;      % Índice para calcular la repetición de frecuencia.

    fprintf('nA continuación se medirán los parámetros del filtro para, posteriormente, sintonizarlo de forma
    automática.\n');
    fprintf('Se le pedirá que realice una serie de acciones. Para pasar a la siguiente acción, pulse cualquier tecla.\n');

    fprintf('t1.-Sitúe el tornillo de ajuste de la frecuencia central del filtro lo más introducido posible en el
    orificio.\n\tPulse cualquier tecla.\n');
    pause;

    fprintf('t2.-Encaje el tornillo del filtro en la puntera del motor.\n\tPulse cualquier tecla.\n');
    pause;

    fprintf('nAhora se procederá a la medición del filtro. Si desea detener el proceso, pulse el botón de PARO.\nPulse
    cualquier tecla para empezar.\n');
    pause;

% Se mide el parámetro S21 para extraer la frecuencia mínima del filtro.
    calcular_S;
    calcular_fc;

    %-----Gráficas-----
    v_fc(indice)=fc;
    v_s21max(indice)=s21max;
    if(indice==1)
        v_diff(indice)=0; % No hay frecuencia anterior a comparar.
    else
        v_diff(indice)=v_fc(indice)-v_fc(indice-1);
    end
    v_WB(indice)=WB;
    indice=indice+1;
    %-----
```

```

ffmin=fc;
fprintf('\nLa frecuencia central mínima sintonizable por el filtro es: %d Hz.\n', ffmin);

if(ffmin<fmin)
    error('MyToolbox:ffmin<Fmin', 'La frecuencia de inicio de medida (Fmin) del Analizador es mayor que la
frecuencia mínima sintonizada por el filtro.');
```

% Se recorre todo el filtro paso a paso para medir la frecuencia superior
% del filtro.

```

Modo='full';
Direccion='izquierda';

while ((fc < fmax) && (repeticion>0))
    fdw=fc;
    envia_orden;
    calcular_S;
    calcular_fc;
    fup=fc;
    %-----Gráficas-----
    v_fc(indice)=fc;
    v_s21max(indice)=s21max;
    v_dif(indice)=v_fc(indice)-v_fc(indice-1);
    v_WB(indice)=WB;
    indice=indice+1;
    %-----
    if (fup==fdw)
        repeticion=repeticion-1
        while ((fc < fmax) && (repeticion>0))
            fdw=fc;
            envia_orden;
            calcular_S;
            calcular_fc;
            fup=fc;
            %-----Gráficas-----
            v_fc(indice)=fc;
            v_s21max(indice)=s21max;
            v_dif(indice)=v_fc(indice)-v_fc(indice-1);
            v_WB(indice)=WB;
            indice=indice+1;
            %-----
            if (fup==fdw)
                repeticion=repeticion-1
            else
                repeticion=6
                break
            end
        end
    end
end
end

% Se extrae la frecuencia máxima del filtro.
ffmax=fc;
fprintf('\nLa frecuencia central máxima sintonizable por el filtro es: %d Hz.\n', ffmax);

% Presentación de los resultados en pantalla
scrz = get(0,'ScreenSize');
figure('Position',[scrz(3)/5 scrz(4)/7 4*scrz(3)/5 6*scrz(4)/7]); title('Parametrización del filtro');
```



```
subplot(4,1,1); plot(v_fc,'r','LineWidth',2); grid on; xlabel('Pasos'); ylabel('Frecuencia Central (Hz)'); title('fc -  
pasos');  
subplot(4,1,2); bar(v_dif,'b'); grid on; xlabel('Pasos'); ylabel('Frecuencia (Hz)'); title('Incremento de frecuencia  
entre pasos');  
subplot(4,1,3); bar(v_WB,'g'); grid on; xlabel('Pasos'); ylabel('Ancho de banda (Hz)'); title('Ancho de banda en  
cada paso');  
subplot(4,1,4); bar(v_s21max,'y'); grid on; xlabel('Pasos'); ylabel('Pérdidas de inserción (dB)'); title('Pérdidas de  
inserción en cada paso');  
  
% Se guardan los parámetros del filtro.  
save(archivo_filtro, 'ffmin', 'ffmax', 'v_fc', 'v_dif', 'v_WB', 'v_s21max');  
  
return
```

Archivo *calcular_S.m*

```
%calcular_S
% Calcula el parámetro S21 del filtro analizado. El formato de la
% matriz es el siguiente: la primera columna se corresponde con la
% frecuencia y la segunda con el S21. Posteriormente, en el programa
% principal, se añade el parámetro S11 en una tercera columna.
%
% La dimensión de las columnas es igual al número de puntos en que se ha
% muestreado el rango fmin - fmax.

function calcular_S

global adress
global freq
global s21
global S

% Se reciben las medidas del Analizador de redes

enviar(adress,'S21;SING;FORM2;OUTPDATA');
[s21]=rebre(adress);

%Se construye una matriz con los parámetros S21
S(:,1)=freq;
S(:,2)=20*log10(abs(s21));

return
```

Archivo *enviar.cpp*

```

/*=====
* enviar.cpp
* Format de crida desde Matlab V.5.2:
*
* enviar(adreça,cadena)
* on
* adreça es la direccio GPIB principal de l'instrument
* cadena es l'ordre a enviar a l'instrument
* -----
* Per compilar desde Matlab
* mex enviar.cpp gpib-32.obj
* Nota: els fitxers
* decl-32.h
* gpib-32.obj
* es tenen que trobar al directori dels *.cpp
* -----
* Autor: A.Lázaro
* (C) TSC-UPC
*=====*/

/* Inclusio de lliberies*/

/*Per Matlab*/

#include "mex.h"

/*Per la tarja GPIB*/
#include <windows.h>
#include "decl-32.h"

/*Per comoditat es defineixen les macros als arguments d'entrada*/

#define IN1 prhs[0]
#define IN2 prhs[1]

//Defines de la tarja
#define BDINDEX 0 // Board Index
#define PRIMARY_ADDR_OF_DMM 1 // Primary address of device
#define NO_SECONDARY_ADDR 0 // Secondary address of device
#define TIMEOUT T10s // Timeout value = 10 seconds
#define EOTMODE 1 // Enable the END message
#define EOSMODE 0 // Disable the EOS mode

/* Declaracio de funcions */

void gpiberr(int ud,char *msg);/*Funcio d'errors*/

/* -----
* FUNCIO PRINCIPAL
* -----*/

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{

```

```

double *dir;
char *ordre;
int N;
char direccio;
int dvm,gpib;

/* Comprobo el nombre de parametres */

if(nrhs != 2)
    mexErrMsgTxt("La funcio necessita dos arguments.");
else
    {
        /*Primer argument es la adreça que es numeric*/
        if( !mxIsNumeric(IN1) )
            mexErrMsgTxt("El primer argument ha d'esser un integer.");
        /*Mira si el segons argument es un string*/
        if( !mxIsChar(IN2) )
            mexErrMsgTxt("El segon argument ha d'esser una cadena.\n");

    }

if(nlhs > 1)
    mexErrMsgTxt("Nombre d'arguments de sortida incorrecte.");

/*Agafo l'orde del segon argument*/
N=mxGetN(IN2)+1;
ordre=(char *)mxCalloc(N,sizeof(char));
mxGetString(IN2,ordre,N);

/*Agafo la direccio del primer argument*/
dir=mxGetPr(IN1);
direccio=(char) dir[0];

    /*Mostro per pantalla les ordres i adreça*/
    /*mexPrintf("Adreça: %d \nOrdre: %s\n",direccio,ordre);*/

    /*Obro port GPIB*/
    /*gpib=ibfind("GPIB0");*/
SendIFC(0);
if (ibsta & ERR)
    {
        gpiberr(BDINDEX,"Board Error");
        return;
    }

gpib=ibfind("GPIB0");
if (ibsta & ERR)
    {
        gpiberr(BDINDEX,"IBFIND Error");
        return;
    }

dvm=ibdev (BDINDEX, direccio, NO_SECONDARY_ADDR,TIMEOUT, EOTMODE, EOSMODE);
if (dvm<0)
    {
        gpiberr(dvm,"ibdev Error");
        return;
    }
else
    {

```

```

        /*Envia l'ordre*/
        ibwrt(dvm,ordre,N-1);
        if (ibsta&ERR) gpiberr(dvm,"ibwrt Error");

        /*Clear dispositiu*/
        /*      ibclr(dvm);
        if (ibsta & ERR) gpiberr(dvm,"ibclr Error");
        */

    }

    /*Tanco Gpib*/
    ibonl(dvm,0);

    /*Allibero memoria*/
    mxFree(ordre);
}

/*-----
* Codi Funcions
*-----*/

/*Rutina de gestio d'error*/

void gpiberr(int ud,char *msg)
{
char *msg_error;

mexPrintf("%s\n",msg);

    if (ibsta & ERR ) msg_error="ERR";
    if (ibsta & TIMO) msg_error="TIMO";
    if (ibsta & END ) msg_error="END";
    if (ibsta & SRQI) msg_error="SRQI";
    if (ibsta & RQS ) msg_error="RQS";
    if (ibsta & CMPL) msg_error="CMPL";
    if (ibsta & LOK ) msg_error="LOK";
    if (ibsta & REM ) msg_error="REM";
    if (ibsta & CIC ) msg_error="CIC";
    if (ibsta & ATN ) msg_error="ATN";
    if (ibsta & TACS) msg_error="TACS";
    if (ibsta & LACS) msg_error="LACS";
    if (ibsta & DTAS) msg_error="DTAS";
    if (ibsta & DCAS) msg_error="DCAS";
    mexPrintf("ibsta=&H%x <%s>\n",ibsta,msg_error);

    if (iberr == EDVR) msg_error="EDVR <DOS Error>";
    if (iberr == ECIC) msg_error="ECIC <Not CIC>";
    if (iberr == ENOL) msg_error="ENOL <No Listener>";
    if (iberr == EADR) msg_error="EADR <Address error>";
    if (iberr == EARG) msg_error="EARG <Invalid argument>";
    if (iberr == ESAC) msg_error="ESAC <Not Sys Ctrlr>";
    if (iberr == EABO) msg_error="EABO <Op. aborted>";
    if (iberr == ENEB) msg_error="ENEB <No GPIB board>";
    if (iberr == EOIP) msg_error="EOIP <Async I/O in prg>";
    if (iberr == ECAP) msg_error="ECAP <No capability>";
    if (iberr == EFSO) msg_error="EFSO <File sys. error>";
    if (iberr == EBUS) msg_error="EBUS <Command error>";
    if (iberr == ESTB) msg_error="ESTB <Status byte lost>";
    if (iberr == ESRQ) msg_error="ESRQ <SRQ stuck on>";

```

```
if (iberr == ETAB) msg_error="ETAB <Table Overflow>";

mexPrintf("iberr= %s\n",msg_error);
mexPrintf("ibcnt= %d\n",ibcntl);
if (ud != -1)
{
    mexPrintf("Cleanup: Taking device off-line\n");
    ibonl (ud, 0);
}
}
```

Archivo *rebbe.cpp*

```

/*=====
* rebbe.cpp
* Format de crida desde Matlab V.5.2:
*
* S=rebbe(adreça)
* on
* adreça es la direccio GPIB principal de l'instrument
* S es una variable de l'entorn MATLAB on es guardara
* el parametre demanat en format complexe tipus FORM2
* -----
* Per compilar desde Matlab
* mex rebbe.cpp gpib-32.obj
* Nota: els fitxers
* decl-32.h
* gpib-32.obj
* es tenen que trobar al directori dels *.cpp
* -----
* Autor: A.Lázaro
* (C) TSC-UPC
*=====*/

/* Inclusio de lliberies*/

/*Per Matlab*/

#include "mex.h"

/*Per la tarja GPIB*/
#include <windows.h>
#include "decl-32.h"

/*Per comoditat es defeneixen les macros als arguments d'entrada/sortida*/

#define IN1 prhs[0]
#define RES plhs[0]

//Defines de la tarja
#define BDINDEX 0 // Board Index
#define PRIMARY_ADDR_OF_DMM 1 // Primary address of device
#define NO_SECONDARY_ADDR 0 // Secondary address of device
#define TIMEOUT T300s // Timeout value = 10 seconds
#define EOTMODE 1 // Enable the END message
#define EOSMODE 0 // Disable the EOS mode

/* Declaracio de funcions */

void gpiberr(int ud,char *msg);/*Funcio d'errors*/
void form2(double *sortida,double *entrada,int npunts); /*Converteix format FORM2 a real del PC*/

/* -----
* FUNCIO PRINCIPAL
* -----*/

void mexFunction( int nlhs, mxArray *plhs[],

```

```

    int nrhs, const mxArray *prhs[] )
{
    mxArray *AUX;
    double *dir,*buffer,*buffei,*aux_real,*aux_imag;
    float *auxr,*auxi;
    char header[2];
    unsigned char ordre[1];
    unsigned long nbytes,npunts,loop;
    unsigned char direccio;
    int dvm,gpib;

    /* Comprobo el nombre de parametres */

    if(nrhs != 1)
        mexErrMsgTxt("La funcio necessita un argument.");
    else
    {
        /*Primer argument es la adreça que es numeric*/
        if( !mxIsNumeric(IN1) )
            mexErrMsgTxt("El primer argument ha d'esser un integer.");
    }

    if(nlhs > 1)
        mexErrMsgTxt("Nombre d'arguments de sortida incorrecte.");

    /*Agafo la direccio del primer argument*/
    dir = mxGetPr(IN1);
    direccio=(unsigned char) dir[0];
    mexPrintf("Adreça: %d\n",direccio);

    /*Obro port GPIB*/
    SendIFC(BDINDEX);
    if (ibsta & ERR)
    {
        gpiberr(BDINDEX,"Board Error");
        return;
    }

    dvm=ibdev (BDINDEX, direccio, NO_SECONDARY_ADDR,TIMEOUT, EOTMODE, EOSMODE);
    if (dvm<0)
    {
        gpiberr(dvm,"ibdev Error");
        return;
    }
    else
    {

    /*Llegim el simbol de capçalera del missatge #A*/

    ibrd(dvm,header,2);
    header[2]='\0';
    mexPrintf("Header = %s\n",header);

    /* Llegim de l'instrument el nombre de bytes que rebrem
    * com que son dos floats (32 real, 32 imag), el número de
    * punts ,s nbytes/8. */

    ibrd(dvm,ordre,2);

```



```

nbytes=ordre[0]*256+ordre[1];
npunts=nbytes/8;

mexPrintf("Nbytes = %d\n",nbytes);
mexPrintf("Npunts = %d\n",npunts);

/*Creem la Matriu de sortida i una matriu auxiliar on possarem
* inicialment les dades llegendes. */

AUX=mxCreateDoubleMatrix(npunts,1,mxCOMPLEX);
aux_real= mxGetPr(AUX);
aux_imag= mxGetPi(AUX);

RES=mxCreateDoubleMatrix(npunts,1,mxCOMPLEX);
buffer = mxGetPr(RES);
bufpei = mxGetPi(RES);

/* Llegeixo de l'instrument un punter amb totes les dades.
* Notar que llegeixo doubles (64 bits), per tant en cada
* posició tenim la part real i la imaginària mesclades.*/

ibrd (dvm,aux_real,nbytes);
if (ibsta & ERR) gpiberr(dvm,"ibwrdr Error");

/* Separem part real i part imaginària utilitzant els punters
* auxiliars auxr i auxi que són de tipus float 32 bites.*/

auxr=(float *)aux_real;
auxi=(float *)auxr+1;
for (loop = 0; loop < npunts; loop++)
{
    aux_imag[loop]=*auxi;
    aux_real[loop]=*auxr;
    auxr+=2;
    auxi+=2;
}

/*Si les dades son en format FORM2 cal intercanviar els bytes
* per tal de que l'estructura de dades sigui compatible amb
* el format per a PC.
* Així canviem el primer per el quart byte i el segon per el tercer./

form2(buffer,aux_real,npunts);
form2(bufpei,aux_imag,npunts);

/*Allibero la memòria reservada per a la matriu
* auxiliar. */

mxDestroyArray(AUX);

/*Clear dispositiu*/
/*ibclr(dvm);
if (ibsta & ERR) gpiberr(dvm,"ibclr Error");
*/

}

/*Tanco Gpib*/
ibonl(dvm,0);

```

```

}

/*-----
* Codi Funcions
*-----*/

/* Funcio form2:
*
* Accio: Realitza la conversi3 de FORM2 a format numeric
*       valid per PC.
*/

void form2(double *sortida,double *entrada,int npunts)
{
    unsigned char tmp,*point1,*point2,*point3,*point4;
    float valor;
    int loop;

    /*Si les dades s3n en format FORM2 cal intercanviar els bytes
    * per tal de que l'estructura de dades sigui compatible amb
    * el format per a PC.
    * Aixi canviem el primer per el quart byte i el segon per el tercer.*/

    for (loop = 0; loop < npunts; loop++)
    {
        valor=(float) entrada[loop];
        point1=(unsigned char *)&valor;
        point2=point1+1;
        point3=point2+1;
        point4=point3+1;
        tmp=*point4;
        *point4=*point1;
        *point1=tmp;
        tmp=*point3;
        *point3=*point2;
        *point2=tmp;
        sortida[loop]= valor;
    }
}

/*Rutina de gestio d'error*/

void gpiberr(int ud,char *msg)
{
    char *msg_error;

    mexPrintf("%s\n",msg);

    if (ibsta & ERR ) msg_error="ERR";
    if (ibsta & TIMO) msg_error="TIMO";
    if (ibsta & END ) msg_error="END";
    if (ibsta & SRQI) msg_error="SRQI";
    if (ibsta & RQS ) msg_error="RQS";
    if (ibsta & CMPL) msg_error="CMPL";
    if (ibsta & LOK ) msg_error="LOK";
    if (ibsta & REM ) msg_error="REM";
    if (ibsta & CIC ) msg_error="CIC";
    if (ibsta & ATN ) msg_error="ATN";
    if (ibsta & TACS) msg_error="TACS";
    if (ibsta & LACS) msg_error="LACS";

```

```

    if (ibsta & DTAS) msg_error="DTAS";
    if (ibsta & DCAS) msg_error="DCAS";
mexPrintf("ibsta=&H%x <%s>\n",ibsta,msg_error);

    if (iberr == EDVR) msg_error="EDVR <DOS Error>";
    if (iberr == ECIC) msg_error="ECIC <Not CIC>";
    if (iberr == ENOL) msg_error="ENOL <No Listener>";
    if (iberr == EADR) msg_error="EADR <Address error>";
    if (iberr == EARG) msg_error="EARG <Invalid argument>";
    if (iberr == ESAC) msg_error="ESAC <Not Sys Ctrlr>";
    if (iberr == EABO) msg_error="EABO <Op. aborted>";
    if (iberr == ENEB) msg_error="ENEB <No GPIB board>";
    if (iberr == EOIP) msg_error="EOIP <Async I/O in prg>";
    if (iberr == ECAP) msg_error="ECAP <No capability>";
    if (iberr == EFSO) msg_error="EFSO <File sys. error>";
    if (iberr == EBUS) msg_error="EBUS <Command error>";
    if (iberr == ESTB) msg_error="ESTB <Status byte lost>";
    if (iberr == ESRQ) msg_error="ESRQ <SRQ stuck on>";
    if (iberr == ETAB) msg_error="ETAB <Table Overflow>";

mexPrintf("iberr= %s\n",msg_error);
mexPrintf("ibcnt= %d\n",ibcntl);
if (ud != -1)
{
    mexPrintf("Cleanup: Taking device off-line\n");
    ibonl (ud, 0);
}
}

```

Archivo *calcular_fc.m*

```
%calcular_fc
% Esta función calcula la frecuencia central del filtro como la frecuencia
% intermedia del ancho de banda a -3dB en la banda de paso.

function calcular_fc

global Npuntos
global S
global fc
global s2lmax
global WB

%Primero se busca la frecuencia en la que se encuentra la máxima
%transferencia de potencia y su valor.
[s2lmax,imax]=max(S(:,2));
fcmax=S(imax,1);

%Se calcula 3dBs por debajo del máximo
db3=((s2lmax)-3);

%Se calcula la frecuencia de 3 dB's por encima
for k=imax:1:Npuntos
    if S(k,2)<=db3
        f3dbup=S(k,1);
        break
    end
end

if isempty(f3dbup)
    error('MyToolbox:NO_f3dbup', 'No se ha encontrado frecuencia de corte superior a 3dB en el margen
especificado.\n')
end

%Se calcula la frecuencia de 3 dB's por abajo
for k=imax:-1:0
    if S(k,2)<=db3
        f3dbdw=S(k,1);
        break
    end
end

if isempty(f3dbdw)
    error('MyToolbox:NO_f3dbdw', 'No se ha encontrado frecuencia de corte inferior a 3dB en el margen
especificado.\n')
end

%Una vez calculadas las frecuencias de corte superior e inferior se calcula
%el ancho de banda y la frecuencia central.
WB=abs(f3dbup-f3dbdw);
fc=((f3dbdw+f3dbup)/2)

return
```

Archivo *calcular_fc_simulacion.m*

```
%calcular_fc_simulacion
% Esta función simula el cálculo de la frecuencia central del filtro. Sirve
% para comprobar el correcto funcionamiento automático del programa sin
% disponer de recursos necesarios como el Analizador de espectros o el
% mismo filtro de microondas.

function calcular_fc_simulacion

global fc
global Modo
global Direccion
global fcv
global puntero_fcv

% Dependiendo de si el motor gira un paso a la derecha o a la izquierda, se
% incrementa o decrementa el puntero del vector. También se tiene en cuenta
% si hace medio paso o un paso, moviéndose 1 o 2 posiciones respectivamente.

switch lower(Modo)
    case 'full'
        switch lower(Direccion)
            case 'izquierda'
                puntero_fcv=puntero_fcv+2;
            case 'derecha'
                puntero_fcv=puntero_fcv-2;
            otherwise

        end

    case 'half'
        switch lower(Direccion)
            case 'izquierda'
                puntero_fcv=puntero_fcv+1;
            case 'derecha'
                puntero_fcv=puntero_fcv-1;
            otherwise

        end

    otherwise

end

% Antes de salir se actualiza el valor de fc.
fc=fcv(puntero_fcv)
return
```

Archivo *envia_orden.m*

```
%envia_orden
% Esta función envia por el puerto serie las ordenes del 'main' teniendo en
% cuenta la dirección (derecha o izquierda) y el modo (full o half)
% transformando estas mismas a un byte comprensible por el PIC.

function envia_orden

global serie
global Modo
global Direccion
global input_RS232

switch lower(Modo)
case 'full'
    switch lower(Direccion)
    case 'izquierda'
        fprintf(serie, '%c', 9);
        input_RS232=fread(serie, 1)
        if input_RS232== 137    %Movimiento OK.
            return
        elseif input_RS232== 105    %Movimiento K.O. por PARO EMERGENCIA!!
            error('MyToolbox:NO_FULL_LEFT', 'No se puede girar un paso a la izquierda el motor por paro
emergencia.\n');
        elseif input_RS232== 80    %Dato no válido. Repetir Tx.
            fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
            repetir=1;
            while (repetir==1)
                fprintf(serie, '%c', 9);
                input_RS232=fread(serie, 1)
                if input_RS232== 137
                    repetir=0;
                elseif input_RS232== 105
                    repetir=0;
                    error('MyToolbox:NO_FULL_LEFT', 'No se puede girar un paso a la izquierda el motor por paro
emergencia.\n');
                elseif input_RS232== 80
                    fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
                else
                    repetir=0;
                    error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
                end
            end
            return
        else
            error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
        end
    case 'derecha'
        fprintf(serie, '%c', 10);
        input_RS232=fread(serie, 1)
        if input_RS232== 138
            return
        elseif input_RS232== 90
            error('MyToolbox:NO_FULL_RIGHT', 'No se puede girar un paso a la derecha el motor por paro
emergencia.\n');
```

```

    elseif input_RS232== 80    %Dato no válido. Repetir Tx.
        fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
        repetir=1;
        while (repetir==1)
            fprintf(serie, '%c', 10);
            input_RS232=fread(serie, 1)
            if input_RS232== 138
                repetir=0;
            elseif input_RS232== 90
                repetir=0;
                error('MyToolbox:NO_FULL_LEFT', 'No se puede girar un paso a la derecha el motor por paro
emergencia.\n');
            elseif input_RS232== 80
                fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
            else
                repetir=0;
                error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
            end
        end
        return
    else
        error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
    end
    otherwise
        error('MyToolbox:DIRECCION_ERROR', 'La variable DIRECCION tiene un valor incorrecto
(Derecha o Izquierda).\n');
    end

case 'half'
    switch lower(Direccion)
        case 'izquierda'
            fprintf(serie, '%c', 5);
            input_RS232=fread(serie, 1)
            if input_RS232== 133
                return
            elseif input_RS232== 101
                error('MyToolbox:NO_HALF_LEFT', 'No se puede girar medio paso a la izquierda el motor por paro
emergencia.\n');
            elseif input_RS232== 80    %Dato no válido. Repetir Tx.
                fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
                repetir=1;
                while (repetir==1)
                    fprintf(serie, '%c', 5);
                    input_RS232=fread(serie, 1)
                    if input_RS232== 133
                        repetir=0;
                    elseif input_RS232== 101
                        repetir=0;
                        error('MyToolbox:NO_FULL_LEFT', 'No se puede girar medio paso a la izquierda el motor por
paro emergencia.\n');
                    elseif input_RS232== 80
                        fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
                    else
                        repetir=0;
                        error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
                    end
                end
                return
            else
                error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
            end
    end

```

```

end
case 'derecha'
    fprintf(serie, '%c', 6);
    input_RS232=fread(serie, 1)
    if input_RS232== 134
        return
    elseif input_RS232== 86
        error('MyToolbox:NO_HALF_RIGHT', 'No se puede girar medio paso a la derecha el motor por paro
emergencia.\n');
    elseif input_RS232== 80    %Dato no válido. Repetir Tx.
        fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
        repetir=1;
        while (repetir==1)
            fprintf(serie, '%c', 6);
            input_RS232=fread(serie, 1)
            if input_RS232== 134
                repetir=0;
            elseif input_RS232== 86
                repetir=0;
                error('MyToolbox:NO_FULL_LEFT', 'No se puede girar medio paso a la derecha el motor por
paro emergencia.\n');
            elseif input_RS232== 80
                fprintf('El PIC ha recibido un dato NO válido. Se repite la Tx...\n');
            else
                repetir=0;
                error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
            end
        end
        return
    else
        error('MyToolbox:INVALID_DATA', 'El PIC ha devuelto un dato NO válido.\n');
    end
    otherwise
        error('MyToolbox:DIRECCION_ERROR', 'La variable DIRECCION tiene un valor incorrecto
(Derecha o Izquierda).\n');
    end

otherwise
    error('MyToolbox:MODO_ERROR', 'La variable MODO tiene un valor incorrecto (Full o Half).\n');
end

```


Archivo *finalizar_puerto.m*

```
%finalizar_puerto
% Función que cierra el puerto serie del PC 'COM1' para que posteriormente
% pueda volver a ser utilizado.

function finalizar_puerto

global serie
global input_RS232

% Primero se comprueba que no hay datos pendientes de leer como errores y
% demás.
if serie.BytesAvailable > 0
    fprintf('\nHay %d datos sin leer en el buffer de entrada\n', serie.BytesAvailable);
    while (serie.BytesAvailable > 0)
        input_RS232=fread(serie, 1) % Se leen los datos y se muestran por pantalla.
    end
end
% Después de leer los datos pendientes, se cierra la comunicación.
fprintf(serie, '%c', 64);
input_RS232=fread(serie, 1)

% Y se comprueba si hay errores al cerrarla.
if input_RS232== 64
    fprintf('La comunicación con el PIC ha concluido satisfactoriamente.\n');
elseif isempty(input_RS232)
    error('MyToolbox:PIC_NO_KO', 'No se puede finalizar la comunicación: El PIC no responde.');
```

else

error('MyToolbox:PIC_NO_TX_KO', 'No se puede finalizar la comunicación: El PIC no responde correctamente.');

end

% **Liberamos el puerto serie** para que pueda ser utilizado por otros threads

% en Matlab.

fclose(serie);

delete(serie);

clear serie;

return

Anexo C.: Código del Firmware

En las siguientes páginas se encuentran los códigos de los archivos componentes del firmware desarrollados en el programa MPLAB de *Microchip* y programados en C.

El orden de los archivos es el siguiente:

- *main.c*
- *PORTS.c*
- *PORTS.h*
- *REGISTERS.c*
- *REGISTERS.h*
- *STATES.c*
- *STATES.h*
- *DELAY.c*
- *DELAY.h*
- *Pic16f72x.h*

Archivo *main.c*

```

#include <pic.h>
#include <htc.h>
#include "PORTS.h"
#include "REGISTERS.h"
#include "STATES.h"
#include "DELAY.h"
#include "pic16f72x.h"

unsigned char RX_WORD;
bit COMMS_OK=0;
bit STEP=0;
int i=5;
unsigned int DIRECTION;
unsigned int MODE;

void main(void){
    INIT_PORTS();
    INIT_REGS();
    INIT_STATE();
    while(1){
        while(COMMS_OK==0){} //Wait to PC is Ready.
        while(COMMS_OK){
            if(STEP) //Main sequence, STEPS.
            {
                PUSH_STATE(DIRECTION, MODE);
                STEP=0;
                ENAC=0; //Disable coils.
                ENBD=0;
            }
        }
    }
}

void interrupt ISR(void)
{
    if(RCIF==1) //RX RS-232 Interrupt.
    {
        RX_WORD=RCREG; //Read data in port RS-232.
        if(RX_WORD==128 && COMMS_OK==0) //Open communication.
        {
            BUCLE_PRAL=1; //LED Lighting -> Go to main sequence.
            COMMS_OK=1; //The communication with PC is ready.
            while(TXIF==0){} //Send to PC if no Tx is waiting:
            TXREG=0x80; //the PIC is ready too.
        }
        else if (RX_WORD==128 && COMMS_OK==1) //The communication is already open
        {
            while(i>0) //Flash LED indicates communication is already open
            {
                BUCLE_PRAL=0;
                DELAY_MS(500);
                BUCLE_PRAL=1;
                DELAY_MS(500);
                i=i-1;
            }
        }
    }
}

```

```

    }
    i=5;
    while(TXIF==0){} //Send to PC if no Tx is waiting:
                      TXREG=0x80; //the PIC is ready too.
}
else if(RX_WORD==64 && COMMS_OK==1) //Close communication.
{
    BUCLE_PRAL=0; //LED no Lighting -> Out of main sequence.
    COMMS_OK=0; //Exit to the main sequence & wait to new connection.
    ENAC=0; //Disable motor coils A and C;
    ENBD=0; //Disable motor coils B and D;
    while(TXIF==0){} //Send to PC if no Tx is waiting:
                      TXREG=0x40; //the PIC close communication.
}
else if (RX_WORD==64 && COMMS_OK==0) //The communication is closed
{
    while(i>0) //Flash LED indicates communication is closed
    {
        BUCLE_PRAL=1;
        DELAY_MS(500);
        BUCLE_PRAL=0;
        DELAY_MS(500);
        i=i-1;
    }
    i=5;
    while(TXIF==0){} //Send to PC if no Tx is waiting:
                      TXREG=0x40; //the PIC close communication.
}
else //The PC send a movement.
{
    DIRECTION=RX_WORD&0x03; //Extract info. DIRECTION.
    MODE=RX_WORD&0x0C; //Extract info. MODE.
    STEP=1; //Do the Step. Unlock main sequence.
}
}

if(RBIF==1) //PORTB Interrupt (inputs).
{
    if(STOP==1) //STOP switch Disabled.
    {
        INFO_0=0; //INFO LED 0 ON.
    }
    if(ONE_STEP==1) //One step left switch Disabled.
    {
        INFO_1=0; //INFO LED 1 OFF.
    }
    if(STOP==0) //STOP switch Enabled.
    {
        INFO_0=1; //INFO LED 0 OFF.
    }
    if(ONE_STEP==0) //One step left switch Enabled.
    {
        if(COMMS_OK==0){ //If no stepping
            INFO_1=1; //INFO LED 1 ON.
            PUSH_ONE_STEP();
        }
    }
}
RBIF=0; //Reset PORTB interrupt flag.
}
}

```

Archivo **PORTS.c**

```
#include <pic.h>
#include <htc.h>
#include "PORTS.h"
#include "REGISTERS.h"
#include "STATES.h"
#include "DELAY.h"
#include "pic16f72x.h"

void INIT_PORTS(void)
{
    //-----PORT A-----
    ANSELA=0x00;           //Set PORTA like Digital I/O
    PORTA=0x00;           //Init PORTA

    TRISA1=0;             //RA1 is an output
    TRISA2=0;             //RA2 is an output
    TRISA3=0;             //RA3 is an output
    TRISA4=0;             //RA4 is an output

    //-----PORT B-----
    ANSELB=0x00;           //Set PORTB like Digital I/O
    PORTB=0x00;           //Init PORTB

    TRISB0=1;             //RB0 is an input
    TRISB1=1;             //RB1 is an input

    RBPU=0;               //Pull-ups are enabled by individual
                        //bits in the WPUB register.
    WPUB0=1;               //WEAK PULL-UP RB0 enabled
    WPUB1=1;               //WEAK PULL-UP RB1 enabled
    RBIE=1;               //Enables the PORTB change interrupt.
    IOCB0=1;               //INTERRUPT-ON-CHANGE RB0 enabled
    IOCB1=1;               //INTERRUPT-ON-CHANGE RB1 enabled
    RBIF=0;               //Port B Change Interrupt Flag Bit.Must
                        //be cleared in software.

    //-----PORT C-----
    PORTC=0x00;           //Init PORTC

    //-----PORT D-----
    ANSELD=0x00;           //Set PORTD like Digital I/O
    PORTD=0x00;           //Init PORTD

    //-----PORT E-----
    ANSELE=0x00;           //Set PORTE like Digital I/O
    PORTE=0x00;           //Init PORTE

    TRISE0=0;             //RE0 is an output
    TRISE1=0;             //RE1 is an output
    TRISE2=0;             //RE2 is an output
}
```

Archivo **PORTS.h**

```
#define ENAC          RA1          //ENABLE A & C
#define AC            RA2          //1-->A; 0-->C
#define BD            RA3          //1-->B; 0-->D
#define ENBD          RA4          //ENABLE B & D

#define STOP          RB1          //SWITCH 1
#define ONE_STEP       RB0         //SWITCH 0

#define INFO_0         RE0          //DEVICE INFO LED 0
#define INFO_1         RE1          //DEVICE INFO LED 1
#define BUCLE_PRAL     RE2          //READY IN MAIN LED

extern void INIT_PORTS(void);
```

Archivo **REGISTERS.c**

```
#include <pic.h>
#include <htc.h>
#include "PORTS.h"
#include "REGISTERS.h"
#include "STATES.h"
#include "DELAY.h"
#include "pic16f72x.h"

void INIT_REGS(void)
{
    //-----OSCILLATOR-----
    IRCF0=1;           //Internal Oscillator set to 16 MHz.
    IRCF1=1;

    //-----INTCON REGISTER (INTERRUPTS) -----
    GIE=1;             //Enables the Global Interrupts.
    PEIE=1;            //Enables the Peripheral Interrupts.

    //-----AUSART -----
    BRGH=1;            //High Baud Rate to reduce Errors.
    SPBRG=103;         //Desired Baud Rate 9.600 bauds.
                        //Others Baud Rate for 16MHz && BRGH=1:
                        //    9600 --> SPBRG=103.
                        //   10.4k --> SPBRG=95.
                        //   19.2k --> SPBRG=51.
                        //   57.6k --> SPBRG=16.

    SPEN=1;            //Enable AUSART.
    SYNC=0;            //Asynchronous Transmission.

    TXEN=1;            //Enable TX AUSART.
                        //To TX a character, write it into TXREG register.
                        //TXIF '0' indicates a character is being TX and
                        //another character is WAITING in TSR register.
                        //We must wait TXIF is set to TX.

    RCIE=1;            //Enable RX Interrupt.
    CREN=1;            //Enable RX AUSART.
                        //To RX a character, read RCSTA register to get
                        //the errors and get the 8 LSb from the RCREG register (DATA).
                        //RCIF '1' indicates a character is waiting in receive buffer.
}
```


Archivo **REGISTERS.h**

```
extern void INIT_REGS(void);
```

Archivo **STATES.c**

```

#include <pic.h>
#include <htc.h>
#include "PORTS.h"
#include "REGISTERS.h"
#include "STATES.h"
#include "DELAY.h"
#include "pic16f72x.h"

unsigned int STATE=0;
int j=5;

void INIT_STATE(void)
{
    STATE=0;
}

void PUSH_ONE_STEP(void)
{
    if(STOP==1)
    {
        STATE=((STATE+2)%8);
        PUSH_OUTPUT(STATE);
        ENAC=0; //Disable motor coils A and C;
        ENBD=0; //Disable motor coils B and D;
    }else{
        while(j>0) //Flash LED indicates STOP is enabled too
        {
            INFO_1=0;
            DELAY_MS(500);
            INFO_1=1;
            DELAY_MS(500);
            j=j-1;
        }
        j=5;
    }
}

void PUSH_STATE (unsigned int DRTN, unsigned int MOD)
{
    switch (MOD)
    {
        case 4: //Half Step
            switch (DRTN)
            {
                case 1: //Half Step Left
                    if(STOP==1) //If There's no ALARMS:
                    {
                        STATE=((STATE+1)%8);
                        PUSH_OUTPUT(STATE); //Rotate motor.
                        while(TXIF==0){} //Send to PC the OK.
                        TXREG=0x85;
                    }else{
                        while(TXIF==0){} //Send to PC the KO!!
                        TXREG=0x65;
                    }
            }
    }
}

```

```

        }
        break;

    case 2: //Half Step Right
        if(STOP==1)
        {
            STATE=((STATE-1)%8);
            PUSH_OUTPUT(STATE);
            while(TXIF==0){} //Send to PC the OK.
            TXREG=0x86;
        } else {
            while(TXIF==0){} //Send to PC the KO!!
            TXREG=0x56;
        }
        break;

    default: // Other cases: repeat instruction
        while(TXIF==0){} //Send to PC the KO!!
        TXREG=0x50;
        break;
    }

break;

case 8: //Full Step
    switch (DRTN)
    {
        case 1: //Full Step Left
            if(STOP==1)
            {
                STATE=((STATE+2)%8);
                PUSH_OUTPUT(STATE);
                while(TXIF==0){} //Send to PC the OK.
                TXREG=0x89;
            } else {
                while(TXIF==0){} //Send to PC the KO!!
                TXREG=0x69;
            }
            break;

        case 2: //Full Step Right
            if(STOP==1)
            {
                STATE=((STATE-2)%8);
                PUSH_OUTPUT(STATE);
                while(TXIF==0){} //Send to PC the OK.
                TXREG=0x8A;
            } else {
                while(TXIF==0){} //Send to PC the KO!!
                TXREG=0x5A;
            }
            break;

        default: // Other cases: repeat instruction
            while(TXIF==0){} //Send to PC the KO!!
            TXREG=0x50;
            break;
    }

break;

```

```

        default:                                // Other cases: repeat instruction
            while(TXIF==0){                      //Send to PC the KO!!
                TXREG=0x50;
                break;
            }
    }

void PUSH_OUTPUT(unsigned int ST)
{
    switch (ST)
    {
        case 0:
            AC=1;
            BD=1;
            ENAC=1;
            ENBD=1;

            DELAY_MS(500);                        //Wait Motor Rotation
            ENAC=0;
            ENBD=0;
            //DELAY_MS(500);                      //Simule time Analyzer read.
            break;

        case 1:
            AC=0;
            BD=1;
            ENAC=0;
            ENBD=1;

            DELAY_MS(500);                        //Wait Motor Rotation
            ENAC=0;
            ENBD=0;
            //DELAY_MS(500);                      //Simule time Analyzer read.
            break;

        case 2:
            AC=0;
            BD=1;
            ENAC=1;
            ENBD=1;

            DELAY_MS(500);                        //Wait Motor Rotation
            ENAC=0;
            ENBD=0;
            //DELAY_MS(500);                      //Simule time Analyzer read.
            break;

        case 3:
            AC=0;
            BD=0;
            ENAC=1;
            ENBD=0;

            DELAY_MS(500);                        //Wait Motor Rotation
            ENAC=0;
            ENBD=0;
            //DELAY_MS(500);                      //Simule time Analyzer read.
            break;
    }
}

```

```
case 4:
    AC=0;
    BD=0;
    ENAC=1;
    ENBD=1;

    DELAY_MS(500);           //Wait Motor Rotation
    ENAC=0;
    ENBD=0;
    //DELAY_MS(500);        //Simule time Analyzer read.
break;

case 5:
    AC=0;
    BD=0;
    ENAC=0;
    ENBD=1;

    DELAY_MS(500);           //Wait Motor Rotation
    ENAC=0;
    ENBD=0;
    //DELAY_MS(500);        //Simule time Analyzer read.
break;

case 6:
    AC=1;
    BD=0;
    ENAC=1;
    ENBD=1;

    DELAY_MS(500);           //Wait Motor Rotation
    ENAC=0;
    ENBD=0;
    //DELAY_MS(500);        //Simule time Analyzer read.
break;

case 7:
    AC=1;
    BD=0;
    ENAC=1;
    ENBD=0;

    DELAY_MS(500);           //Wait Motor Rotation
    ENAC=0;
    ENBD=0;
    //DELAY_MS(500);        //Simule time Analyzer read.
break;
}
}
```

Archivo **STATES.h**

```
extern void INIT_STATE (void);  
extern void PUSH_ONE_STEP(void);  
extern void PUSH_STATE (unsigned int DRTN, unsigned int MOD);  
extern void PUSH_OUTPUT (unsigned int ST);
```

Archivo *DELAY.c*

```
#include <pic.h>
#include <htc.h>
#include "PORTS.h"
#include "REGISTERS.h"
#include "STATES.h"
#include "DELAY.h"
#include "pic16f72x.h"

void DELAY_US (unsigned char time_us)
{
    unsigned long us=8*time_us;           //At 16MHz: Tnop=0.125 us --> 1us=8*Tnop
    while (us-->0)
    {
        asm("nop");
    }
}

void DELAY_MS (unsigned char time_ms)
{
    unsigned long us=1000*8*time_ms;      //At 16MHz: Tnop=0.125 us --> 1ms=1000*8*Tnop
    while (us-->0)
    {
        asm("nop");
    }
}
```

Archivo *DELAY.h*

```
extern void DELAY_US (unsigned char us);  
extern void DELAY_MS (unsigned char ms);
```


Archivo *pic16f72x.h*

```
#ifndef _HTC_H_
#warning Header file pic16f72x.h included directly.
#Sse #include <htc.h> instead.
#endif

/* header file for the MICROCHIP PIC microcontroller
16F722
16F723
16F724
16F726
16F727
16LF722
16LF723
16LF724
16LF726
16LF727

*/

#ifndef _PIC16F72X_H
#define _PIC16F72X_H

// Special function register definitions

volatile unsigned char INDF @ 0x000;
volatile unsigned char TMR0 @ 0x001;
volatile unsigned char PCL @ 0x002;
volatile unsigned char STATUS @ 0x003;
volatile unsigned char FSR @ 0x004;
volatile unsigned char PORTA @ 0x005;
volatile unsigned char PORTB @ 0x006;
volatile unsigned char PORTC @ 0x007;
#if defined(_16F724) || defined(_16F727) || \
defined(_16LF724) || defined(_16LF727)
volatile unsigned char PORTD @ 0x008;
#endif
volatile unsigned char PORTE @ 0x009;
volatile unsigned char PCLATH @ 0x00A;
volatile unsigned char INTCON @ 0x00B;
volatile unsigned char PIR1 @ 0x00C;
volatile unsigned char PIR2 @ 0x00D;
volatile unsigned char TMR1L @ 0x00E;
volatile unsigned char TMR1H @ 0x00F;
volatile unsigned char T1CON @ 0x010;
volatile unsigned char TMR2 @ 0x011;
volatile unsigned char T2CON @ 0x012;
volatile unsigned char SSPBUF @ 0x013;
volatile unsigned char SSPCON @ 0x014;
volatile unsigned char CCPR1L @ 0x015;
volatile unsigned char CCPR1H @ 0x016;
volatile unsigned char CCP1CON @ 0x017;
volatile unsigned char RCSTA @ 0x018;
volatile unsigned char TXREG @ 0x019;
volatile unsigned char RCREG @ 0x01A;
volatile unsigned char CCPR2L @ 0x01B;
volatile unsigned char CCPR2H @ 0x01C;
volatile unsigned char CCP2CON @ 0x01D;
volatile unsigned char ADRES @ 0x01E;
volatile unsigned char ADCON0 @ 0x01F;
volatile unsigned char OPTION @ 0x081;
volatile unsigned char TRISA @ 0x085;
```

```
volatile unsigned char TRISB @ 0x086;
volatile unsigned char TRISC @ 0x087;
#if defined(_16F724) || defined(_16F727) || \
defined(_16LF724) || defined(_16LF727)
volatile unsigned char TRISD @ 0x088;
#endif
volatile unsigned char TRISE @ 0x089;
volatile unsigned char PIE1 @ 0x08C;
volatile unsigned char PIE2 @ 0x08D;
volatile unsigned char PCON @ 0x08E;
volatile unsigned char T1GCON @ 0x08F;
volatile unsigned char OSCCON @ 0x090;
volatile unsigned char OSTUNE @ 0x091;
volatile unsigned char PR2 @ 0x092;
volatile unsigned char SSPADD @ 0x093;

// Alternate function
volatile unsigned char SSPMSK @ 0x093;
volatile unsigned char SSPSTAT @ 0x094;
volatile unsigned char WPUB @ 0x095;
volatile unsigned char IOCB @ 0x096;
volatile unsigned char TXSTA @ 0x098;
volatile unsigned char SPBRG @ 0x099;
volatile unsigned char APFCON @ 0x09C;
volatile unsigned char FVRCON @ 0x09D;
volatile unsigned char ADCON1 @ 0x09F;
volatile unsigned char CPSCON0 @ 0x108;
volatile unsigned char CPSCON1 @ 0x109;
volatile unsigned char PMDATL @ 0x10C;

// Alternate definition
volatile unsigned char EEDATA @ 0x10C;
volatile unsigned char PMADRL @ 0x10D;

// Alternate definition
volatile unsigned char EEADR @ 0x10D;
volatile unsigned char PMDATH @ 0x10E;

// Alternate definition
volatile unsigned char EEDATH @ 0x10E;
volatile unsigned char PMADRH @ 0x10F;

// Alternate definition
volatile unsigned char EEADRH @ 0x10F;
volatile unsigned char ANSELA @ 0x185;
volatile unsigned char ANSELB @ 0x186;

#if defined(_16F724) || defined(_16F727) || \
defined(_16LF724) || defined(_16LF727)
volatile unsigned char ANSELD @ 0x188;
volatile unsigned char ANSELE @ 0x189;
#endif

volatile unsigned char PMCON1 @ 0x18C;

/* Definitions for STATUS register */
volatile bit CARRY @ ((unsigned)&STATUS*8)+0;
volatile bit DC @ ((unsigned)&STATUS*8)+1;
volatile bit ZERO @ ((unsigned)&STATUS*8)+2;
volatile bit PD @ ((unsigned)&STATUS*8)+3;
volatile bit TO @ ((unsigned)&STATUS*8)+4;
volatile bit RP0 @ ((unsigned)&STATUS*8)+5;
volatile bit RP1 @ ((unsigned)&STATUS*8)+6;
volatile bit IRP @ ((unsigned)&STATUS*8)+7;

/* Definitions for PORTA register */
```

```
volatile bit RA0 @ ((unsigned)&PORTA*8)+0;
volatile bit RA1 @ ((unsigned)&PORTA*8)+1;
volatile bit RA2 @ ((unsigned)&PORTA*8)+2;
volatile bit RA3 @ ((unsigned)&PORTA*8)+3;
volatile bit RA4 @ ((unsigned)&PORTA*8)+4;
volatile bit RA5 @ ((unsigned)&PORTA*8)+5;
volatile bit RA6 @ ((unsigned)&PORTA*8)+6;
volatile bit RA7 @ ((unsigned)&PORTA*8)+7;
```

```
/* Definitions for PORTB register */
```

```
volatile bit RB0 @ ((unsigned)&PORTB*8)+0;
volatile bit RB1 @ ((unsigned)&PORTB*8)+1;
volatile bit RB2 @ ((unsigned)&PORTB*8)+2;
volatile bit RB3 @ ((unsigned)&PORTB*8)+3;
volatile bit RB4 @ ((unsigned)&PORTB*8)+4;
volatile bit RB5 @ ((unsigned)&PORTB*8)+5;
volatile bit RB6 @ ((unsigned)&PORTB*8)+6;
volatile bit RB7 @ ((unsigned)&PORTB*8)+7;
```

```
/* Definitions for PORTC register */
```

```
volatile bit RC0 @ ((unsigned)&PORTC*8)+0;
volatile bit RC1 @ ((unsigned)&PORTC*8)+1;
volatile bit RC2 @ ((unsigned)&PORTC*8)+2;
volatile bit RC3 @ ((unsigned)&PORTC*8)+3;
volatile bit RC4 @ ((unsigned)&PORTC*8)+4;
volatile bit RC5 @ ((unsigned)&PORTC*8)+5;
volatile bit RC6 @ ((unsigned)&PORTC*8)+6;
volatile bit RC7 @ ((unsigned)&PORTC*8)+7;
```

```
#if defined(_16F724) || defined(_16F727) ||
defined(_16LF724) || defined(_16LF727)
```

```
/* Definitions for PORTD register */
```

```
volatile bit RD0 @ ((unsigned)&PORTD*8)+0;
volatile bit RD1 @ ((unsigned)&PORTD*8)+1;
volatile bit RD2 @ ((unsigned)&PORTD*8)+2;
volatile bit RD3 @ ((unsigned)&PORTD*8)+3;
volatile bit RD4 @ ((unsigned)&PORTD*8)+4;
volatile bit RD5 @ ((unsigned)&PORTD*8)+5;
volatile bit RD6 @ ((unsigned)&PORTD*8)+6;
volatile bit RD7 @ ((unsigned)&PORTD*8)+7;
```

```
/* Definitions for PORTE register */
```

```
volatile bit RE0 @ ((unsigned)&PORTE*8)+0;
volatile bit RE1 @ ((unsigned)&PORTE*8)+1;
volatile bit RE2 @ ((unsigned)&PORTE*8)+2;
#endif
volatile bit RE3 @ ((unsigned)&PORTE*8)+3;
```

```
/* Definitions for INTCON register */
```

```
volatile bit RBIF @ ((unsigned)&INTCON*8)+0;
// Alternate definition for backward compatibility
volatile bit RABIF @ ((unsigned)&INTCON*8)+0;
volatile bit INTF @ ((unsigned)&INTCON*8)+1;
volatile bit T0IF @ ((unsigned)&INTCON*8)+2;
volatile bit RBIE @ ((unsigned)&INTCON*8)+3;
// Alternate definition for backward compatibility
volatile bit RABIE @ ((unsigned)&INTCON*8)+3;
volatile bit INTE @ ((unsigned)&INTCON*8)+4;
volatile bit TOIE @ ((unsigned)&INTCON*8)+5;
volatile bit PEIE @ ((unsigned)&INTCON*8)+6;
volatile bit GIE @ ((unsigned)&INTCON*8)+7;
```

```
/* Definitions for PIR1 register */
```

```
volatile bit TMR1IF @ ((unsigned)&PIR1*8)+0;
volatile bit TMR2IF @ ((unsigned)&PIR1*8)+1;
volatile bit CCP1IF @ ((unsigned)&PIR1*8)+2;
volatile bit SSPIF @ ((unsigned)&PIR1*8)+3;
```

```
volatile bit TXIF @ ((unsigned)&PIR1*8)+4;
volatile bit RCIF @ ((unsigned)&PIR1*8)+5;
volatile bit ADIF @ ((unsigned)&PIR1*8)+6;
volatile bit TMR1GI @ ((unsigned)&PIR1*8)+7;
```

```
/* Definitions for PIR2 register */
```

```
volatile bit CCP2IF @ ((unsigned)&PIR2*8)+0;
```

```
/* Definitions for T1CON register */
```

```
bit TMR1ON @ ((unsigned)&T1CON*8)+0;
bit T1SYNC @ ((unsigned)&T1CON*8)+2;
bit T1OSCEN @ ((unsigned)&T1CON*8)+3;
bit T1CKPS0 @ ((unsigned)&T1CON*8)+4;
bit T1CKPS1 @ ((unsigned)&T1CON*8)+5;
bit TMR1CS0 @ ((unsigned)&T1CON*8)+6;
bit TMR1CS1 @ ((unsigned)&T1CON*8)+7;
```

```
/* Definitions for T2CON register */
```

```
bit T2CKPS0 @ ((unsigned)&T2CON*8)+0;
bit T2CKPS1 @ ((unsigned)&T2CON*8)+1;
bit TMR2ON @ ((unsigned)&T2CON*8)+2;
bit TOUTPS0 @ ((unsigned)&T2CON*8)+3;
bit TOUTPS1 @ ((unsigned)&T2CON*8)+4;
bit TOUTPS2 @ ((unsigned)&T2CON*8)+5;
bit TOUTPS3 @ ((unsigned)&T2CON*8)+6;
```

```
/* Definitions for SSPCON register */
```

```
bit SSPM0 @ ((unsigned)&SSPCON*8)+0;
bit SSPM1 @ ((unsigned)&SSPCON*8)+1;
bit SSPM2 @ ((unsigned)&SSPCON*8)+2;
bit SSPM3 @ ((unsigned)&SSPCON*8)+3;
bit CKP @ ((unsigned)&SSPCON*8)+4;
bit SSPEN @ ((unsigned)&SSPCON*8)+5;
volatile bit SSPOV @ ((unsigned)&SSPCON*8)+6;
volatile bit WCOL @ ((unsigned)&SSPCON*8)+7;
```

```
/* Definitions for CCP1CON register */
```

```
bit CCP1M0 @ ((unsigned)&CCP1CON*8)+0;
bit CCP1M1 @ ((unsigned)&CCP1CON*8)+1;
bit CCP1M2 @ ((unsigned)&CCP1CON*8)+2;
bit CCP1M3 @ ((unsigned)&CCP1CON*8)+3;
bit DC1B0 @ ((unsigned)&CCP1CON*8)+4;
bit DC1B1 @ ((unsigned)&CCP1CON*8)+5;
```

```
/* Definitions for RCSTA register */
```

```
volatile bit RX9D @ ((unsigned)&RCSTA*8)+0;
volatile bit OERR @ ((unsigned)&RCSTA*8)+1;
volatile bit FERR @ ((unsigned)&RCSTA*8)+2;
bit ADDEN @ ((unsigned)&RCSTA*8)+3;
bit CREN @ ((unsigned)&RCSTA*8)+4;
bit SREN @ ((unsigned)&RCSTA*8)+5;
bit RX9 @ ((unsigned)&RCSTA*8)+6;
bit SPEN @ ((unsigned)&RCSTA*8)+7;
```

```
/* Definitions for CCP2CON register */
```

```
bit CCP2M0 @ ((unsigned)&CCP2CON*8)+0;
bit CCP2M1 @ ((unsigned)&CCP2CON*8)+1;
bit CCP2M2 @ ((unsigned)&CCP2CON*8)+2;
bit CCP2M3 @ ((unsigned)&CCP2CON*8)+3;
bit DC2B0 @ ((unsigned)&CCP2CON*8)+4;
bit DC2B1 @ ((unsigned)&CCP2CON*8)+5;
```

```
/* Definitions for ADCON0 register */
```

```
bit ADON @ ((unsigned)&ADCON0*8)+0;
volatile bit GODONE @ ((unsigned)&ADCON0*8)+1;
// compatibility with old devices
volatile bit ADGO @ ((unsigned)&ADCON0*8)+1;
```

```

bit    CHS0    @ ((unsigned)&ADCON0*8)+2;
bit    CHS1    @ ((unsigned)&ADCON0*8)+3;
bit    CHS2    @ ((unsigned)&ADCON0*8)+4;
bit    CHS3    @ ((unsigned)&ADCON0*8)+5;

/* Definitions for OPTION register */
bit    PS0     @ ((unsigned)&OPTION*8)+0;
bit    PS1     @ ((unsigned)&OPTION*8)+1;
bit    PS2     @ ((unsigned)&OPTION*8)+2;
bit    PSA     @ ((unsigned)&OPTION*8)+3;
bit    T0SE    @ ((unsigned)&OPTION*8)+4;
bit    T0CS    @ ((unsigned)&OPTION*8)+5;
bit    INTEDG  @ ((unsigned)&OPTION*8)+6;
bit    RBPU    @ ((unsigned)&OPTION*8)+7;
// Alternate definition for backward compatibility
bit    RABPU   @ ((unsigned)&OPTION*8)+7;

/* Definitions for TRISA register */
bit    TRISA0  @ ((unsigned)&TRISA*8)+0;
bit    TRISA1  @ ((unsigned)&TRISA*8)+1;
bit    TRISA2  @ ((unsigned)&TRISA*8)+2;
bit    TRISA3  @ ((unsigned)&TRISA*8)+3;
bit    TRISA4  @ ((unsigned)&TRISA*8)+4;
bit    TRISA5  @ ((unsigned)&TRISA*8)+5;
bit    TRISA6  @ ((unsigned)&TRISA*8)+6;
bit    TRISA7  @ ((unsigned)&TRISA*8)+7;

/* Definitions for TRISB register */
volatile bit    TRISB0  @ ((unsigned)&TRISB*8)+0;
volatile bit    TRISB1  @ ((unsigned)&TRISB*8)+1;
volatile bit    TRISB2  @ ((unsigned)&TRISB*8)+2;
volatile bit    TRISB3  @ ((unsigned)&TRISB*8)+3;
volatile bit    TRISB4  @ ((unsigned)&TRISB*8)+4;
volatile bit    TRISB5  @ ((unsigned)&TRISB*8)+5;
volatile bit    TRISB6  @ ((unsigned)&TRISB*8)+6;
volatile bit    TRISB7  @ ((unsigned)&TRISB*8)+7;

/* Definitions for TRISC register */
volatile bit    TRISC0  @ ((unsigned)&TRISC*8)+0;
volatile bit    TRISC1  @ ((unsigned)&TRISC*8)+1;
volatile bit    TRISC2  @ ((unsigned)&TRISC*8)+2;
volatile bit    TRISC3  @ ((unsigned)&TRISC*8)+3;
volatile bit    TRISC4  @ ((unsigned)&TRISC*8)+4;
volatile bit    TRISC5  @ ((unsigned)&TRISC*8)+5;
volatile bit    TRISC6  @ ((unsigned)&TRISC*8)+6;
volatile bit    TRISC7  @ ((unsigned)&TRISC*8)+7;

#if defined(_16F724) || defined(_16F727) || \
    defined(_16LF724) || defined(_16LF727)
/* Definitions for TRISD register */
volatile bit    TRISD0  @ ((unsigned)&TRISD*8)+0;
volatile bit    TRISD1  @ ((unsigned)&TRISD*8)+1;
volatile bit    TRISD2  @ ((unsigned)&TRISD*8)+2;
volatile bit    TRISD3  @ ((unsigned)&TRISD*8)+3;
volatile bit    TRISD4  @ ((unsigned)&TRISD*8)+4;
volatile bit    TRISD5  @ ((unsigned)&TRISD*8)+5;
volatile bit    TRISD6  @ ((unsigned)&TRISD*8)+6;
volatile bit    TRISD7  @ ((unsigned)&TRISD*8)+7;

/* Definitions for TRISE register */
volatile bit    TRISE0  @ ((unsigned)&TRISE*8)+0;
volatile bit    TRISE1  @ ((unsigned)&TRISE*8)+1;
volatile bit    TRISE2  @ ((unsigned)&TRISE*8)+2;
#endif
volatile bit    TRISE3  @ ((unsigned)&TRISE*8)+3;

/* Definitions for PIE1 register */
bit    TMR1IE  @ ((unsigned)&PIE1*8)+0;
bit    TMR2IE  @ ((unsigned)&PIE1*8)+1;
bit    CCP1IE  @ ((unsigned)&PIE1*8)+2;
bit    SSPIE   @ ((unsigned)&PIE1*8)+3;
bit    TXIE    @ ((unsigned)&PIE1*8)+4;
bit    RCIE    @ ((unsigned)&PIE1*8)+5;
bit    ADIE    @ ((unsigned)&PIE1*8)+6;
volatile bit    TMR1GIE @ ((unsigned)&PIE1*8)+7;

/* Definitions for PIE2 register */
bit    CCP2IE  @ ((unsigned)&PIE2*8)+0;

/* Definitions for PCON register */
volatile bit    BOR    @ ((unsigned)&PCON*8)+0;
volatile bit    POR    @ ((unsigned)&PCON*8)+1;

/* Definitions for T1GCON register */
bit    T1GSS0  @ ((unsigned)&T1GCON*8)+0;
bit    T1GSS1  @ ((unsigned)&T1GCON*8)+1;
volatile bit    T1GVAL @ ((unsigned)&T1GCON*8)+2;
volatile bit    T1GGO  @ ((unsigned)&T1GCON*8)+3;
bit    T1GSPM  @ ((unsigned)&T1GCON*8)+4;
bit    T1GTM   @ ((unsigned)&T1GCON*8)+5;
bit    T1GPOL  @ ((unsigned)&T1GCON*8)+6;
bit    TMR1GE  @ ((unsigned)&T1GCON*8)+7;

/* Definitions for OSCCON register */
bit    ICSS    @ ((unsigned)&OSCCON*8)+2;
bit    ICSL    @ ((unsigned)&OSCCON*8)+3;
bit    IRCF0   @ ((unsigned)&OSCCON*8)+4;
bit    IRCF1   @ ((unsigned)&OSCCON*8)+5;

/* Definitions for OSCTUNE register */
bit    TUN0    @ ((unsigned)&OSCTUNE*8)+0;
bit    TUN1    @ ((unsigned)&OSCTUNE*8)+1;
bit    TUN2    @ ((unsigned)&OSCTUNE*8)+2;
bit    TUN3    @ ((unsigned)&OSCTUNE*8)+3;
bit    TUN4    @ ((unsigned)&OSCTUNE*8)+4;
bit    TUN5    @ ((unsigned)&OSCTUNE*8)+5;

/* Definitions for SSPSTAT register */
volatile bit    BF     @ ((unsigned)&SSPSTAT*8)+0;
volatile bit    UA     @ ((unsigned)&SSPSTAT*8)+1;
volatile bit    RW     @ ((unsigned)&SSPSTAT*8)+2;
volatile bit    START  @ ((unsigned)&SSPSTAT*8)+3;
volatile bit    STOP   @ ((unsigned)&SSPSTAT*8)+4;
volatile bit    DA     @ ((unsigned)&SSPSTAT*8)+5;
bit    CKE      @ ((unsigned)&SSPSTAT*8)+6;
bit    SMP      @ ((unsigned)&SSPSTAT*8)+7;

/* Definitions for WPUB register */
bit    WPUB0   @ ((unsigned)&WPUB*8)+0;
bit    WPUB1   @ ((unsigned)&WPUB*8)+1;
bit    WPUB2   @ ((unsigned)&WPUB*8)+2;
bit    WPUB3   @ ((unsigned)&WPUB*8)+3;
bit    WPUB4   @ ((unsigned)&WPUB*8)+4;
bit    WPUB5   @ ((unsigned)&WPUB*8)+5;
bit    WPUB6   @ ((unsigned)&WPUB*8)+6;
bit    WPUB7   @ ((unsigned)&WPUB*8)+7;

/* Definitions for IOCB register */
bit    IOCB0   @ ((unsigned)&IOCB*8)+0;
bit    IOCB1   @ ((unsigned)&IOCB*8)+1;
bit    IOCB2   @ ((unsigned)&IOCB*8)+2;
bit    IOCB3   @ ((unsigned)&IOCB*8)+3;
bit    IOCB4   @ ((unsigned)&IOCB*8)+4;
bit    IOCB5   @ ((unsigned)&IOCB*8)+5;

```

```

bit      IOCB6   @ ((unsigned)&IOCB*8)+6;
bit      IOCB7   @ ((unsigned)&IOCB*8)+7;

/* Definitions for TXSTA register */
volatile bit TX9D   @ ((unsigned)&TXSTA*8)+0;
volatile bit TRMT   @ ((unsigned)&TXSTA*8)+1;
bit      BRGH   @ ((unsigned)&TXSTA*8)+2;
bit      SYNC   @ ((unsigned)&TXSTA*8)+4;
bit      TXEN   @ ((unsigned)&TXSTA*8)+5;
bit      TX9     @ ((unsigned)&TXSTA*8)+6;
bit      CSRC   @ ((unsigned)&TXSTA*8)+7;

/* Definitions for SPBRG register */
bit      BRG0   @ ((unsigned)&SPBRG*8)+0;
bit      BRG1   @ ((unsigned)&SPBRG*8)+1;
bit      BRG2   @ ((unsigned)&SPBRG*8)+2;
bit      BRG3   @ ((unsigned)&SPBRG*8)+3;
bit      BRG4   @ ((unsigned)&SPBRG*8)+4;
bit      BRG5   @ ((unsigned)&SPBRG*8)+5;
bit      BRG6   @ ((unsigned)&SPBRG*8)+6;
bit      BRG7   @ ((unsigned)&SPBRG*8)+7;

/* Definitions for APFCON register */
bit      CCP2SEL @ ((unsigned)&APFCON*8)+0;
bit      SSSEL   @ ((unsigned)&APFCON*8)+1;

/* Definitions for FVRCON register */
bit      ADFVR0 @ ((unsigned)&FVRCON*8)+0;
bit      ADFVR1 @ ((unsigned)&FVRCON*8)+1;
volatile bit FVREN @ ((unsigned)&FVRCON*8)+6;
volatile bit FVRRDY @ ((unsigned)&FVRCON*8)+7;

/* Definitions for ADCON1 register */
bit      ADREF0 @ ((unsigned)&ADCON1*8)+0;
bit      ADREF1 @ ((unsigned)&ADCON1*8)+1;
bit      ADCS0  @ ((unsigned)&ADCON1*8)+4;
bit      ADCS1  @ ((unsigned)&ADCON1*8)+5;
bit      ADCS2  @ ((unsigned)&ADCON1*8)+6;

/* Definitions for CPSCON0 register */
bit      TOXCS  @ ((unsigned)&CPSCON0*8)+0;
volatile bit CPSOUT @ ((unsigned)&CPSCON0*8)+1;
bit      CPSRNG0 @ ((unsigned)&CPSCON0*8)+2;
bit      CPSRNG1 @ ((unsigned)&CPSCON0*8)+3;
bit      CPSON   @ ((unsigned)&CPSCON0*8)+7;

/* Definitions for CPSCON1 register */
bit      CPSCH0 @ ((unsigned)&CPSCON1*8)+0;
bit      CPSCH1 @ ((unsigned)&CPSCON1*8)+1;
bit      CPSCH2 @ ((unsigned)&CPSCON1*8)+2;
bit      CPSCH3 @ ((unsigned)&CPSCON1*8)+3;

/* Definitions for ANSELA register */
bit      ANSA0  @ ((unsigned)&ANSELA*8)+0;
bit      ANSA1  @ ((unsigned)&ANSELA*8)+1;
bit      ANSA2  @ ((unsigned)&ANSELA*8)+2;
bit      ANSA3  @ ((unsigned)&ANSELA*8)+3;
bit      ANSA4  @ ((unsigned)&ANSELA*8)+4;
bit      ANSA5  @ ((unsigned)&ANSELA*8)+5;

/* Definitions for ANSELB register */
bit      ANSB0  @ ((unsigned)&ANSELB*8)+0;
bit      ANSB1  @ ((unsigned)&ANSELB*8)+1;
bit      ANSB2  @ ((unsigned)&ANSELB*8)+2;
bit      ANSB3  @ ((unsigned)&ANSELB*8)+3;
bit      ANSB4  @ ((unsigned)&ANSELB*8)+4;
bit      ANSB5  @ ((unsigned)&ANSELB*8)+5;

#if defined(_16F724) || defined(_16F727) || \
    defined(_16LF724) || defined(_16LF727)
/* Definitions for ANSELD register */
bit      ANSD0  @ ((unsigned)&ANSELD*8)+0;
bit      ANSD1  @ ((unsigned)&ANSELD*8)+1;
bit      ANSD2  @ ((unsigned)&ANSELD*8)+2;
bit      ANSD3  @ ((unsigned)&ANSELD*8)+3;
bit      ANSD4  @ ((unsigned)&ANSELD*8)+4;
bit      ANSD5  @ ((unsigned)&ANSELD*8)+5;
bit      ANSD6  @ ((unsigned)&ANSELD*8)+6;
bit      ANSD7  @ ((unsigned)&ANSELD*8)+7;

/* Definitions for ANSELE register */
bit      ANSE0  @ ((unsigned)&ANSELE*8)+0;
bit      ANSE1  @ ((unsigned)&ANSELE*8)+1;
bit      ANSE2  @ ((unsigned)&ANSELE*8)+2;
#endif

/* Definitions for PMCON1 register */
volatile bit RD   @ ((unsigned)&PMCON1*8)+0;

// Configuration Mask Definitions
#define CONFIG_ADDR 0x2007
// Oscillator configurations
#define RCCLKO      0x3FFF
#define RCIO        0x3FFE
#define INTCLKO     0x3FFD
#define INTIO       0x3FFC
#define EC          0x3FFB
#define HS          0x3FFA
#define XT          0x3FF9
#define LP          0x3FF8
// Watchdog timer enable
#define WDTEN       0x3FFF
#define WDTDIS      0x3FF7
// Power up timer enable
#define PWRTEN      0x3FEF
#define PWRTDIS     0x3FFF
// MCLR pin function
#define MCLREN      0x3FFF
#define MCLRDIS     0x3FDF
// Protection of flash memory
#define PROTECT     0x3FBF
#define UNPROTECT   0x3FFF
// Brown out reset enable
#define BOREN       0x3FFF
#define BOREN_XSLP  0x3EFF
#define BORDIS      0x3CFF
// Brown out reset voltage
#define BORV25      0x3BFF
#define BORV19      0x3FFF
// INTOSC PLL enable
#define PLEN        0x3FFF
#define PLLDIS      0x2FFF
// Debugger enable
#define DEBUGEN     0x1FFF
#define DEBUGDIS    0x3FFF
#define CONFIG_ADDR2 0x2008
// Voltage regulator capacitor enable
// - this setting ignored in 16LF devices
#define VCAPRA0     0x3FCF
#define VCAPRA5     0x3FDF
#define VCAPRA6     0x3FEF
#define VCAPDIS     0x3FFF
#endif

```


Anexo D.: BOM

Tabla 17: BOM (*Bill Of Materials*) del hardware implementado.

Designator	Comment	Description	Qty.
C1, C12, C13	100nF	Capacitor	3
C2	1000uF/25V	Polarized Capacitor (Radial)	1
C3	100uF/16V	Polarized Capacitor (Radial)	1
C4	0.1uF	Capacitor	1
C5	10uF/25V	Polarized Capacitor (Radial)	1
C6, C7, C8, C9, C10,	1uF/63V	Polarized Capacitor (Radial)	6
D1, D3, D4	1N4007	1 Amp General Purpose Rectifier	3
D2	POWER	Typical GREEN GaAs LED	1
D5	LED 0	Typical GREEN GaAs LED	1
D6	LED 1	Typical GREEN GaAs LED	1
D7	Bucle Pral.	Typical GREEN GaAs LED	1
D8, D9, D10, D11	STEP	Typical RED GaAs LED	4
D12, D13, D14, D15	1N4933	1 Amp General Purpose Rectifier	4
F1, F2, F3	Fuse 1, 2, 3	Fuse	3
J1	PWR2.5	Low Voltage Power Supply Connector	1
J2	DB Connector 9	Receptacle Assembly, 9 Position, Right Angle	1
J3	RJ-12	Standard Profile, Flush Mount PCB Jack	1
P1	Banana Connector 12V	Plug	1
P2	Banana Connector 5V	Plug	1
P3	Banana Connector Vreg	Plug	1
P4	Power Switch	Switch, 3-Pin	1
P5	Banana Connector GND	Plug	1
P6, P7, P8	Jumper	Header, 3-Pin	3
P9, P10, P11, P12	UC3710T	High Current FET Driver, 1A	4
P13	MOTOR_CONN	Header, 5-Pin	1
P14	SENSOR_CONN	Header, 3-Pin	1
P15	Pin out	Plug	1
P16	Pin 1-20	Header, 20-Pin	1
P17	Pin 21-40	Header, 20-Pin	1
R1, R9, R10, R11, R12, R13, R14, R15	330	Resistor	8
R2, R6	270	Resistor	2
R3	5K	Potentiometer	1
R4	10K	Resistor	1
R5	100	Resistor	1
R7	10K / 1%	Resistor	1
R8	24K / 1%	Resistor	1
R16, R17, R18, R19	470	Resistor	4
R20, R21, R22, R23	680	Resistor	4
S1	Switch	Double-Pole, Single-Throw Switch	1
U1	L7805CV	Positive Voltage Regulator, 5V DC	1
U2	LM317T	1.2V to 37V Voltage Regulator	1
U3	TL431	Adjustable Precision Shunt Regulator	1
U4	MAX232ACPE	+5V Powered, RS-232 Driver/Receiver	1
U5	PIC16F727-I/P	Enhanced FLASH Microcontroller, 40-Pin PDIP	1
U6	SN74LS04N	Hex Inverter - NOT Gate	1
U7	SN74LS08N	Quadruple 2-Input Positive-AND Gate	1
U8, U9, U10, U11	4N27	Optocoupler, Phototransistor Output, TTL	4

Anexo E.: Layouts y disposición de los componentes en el hardware

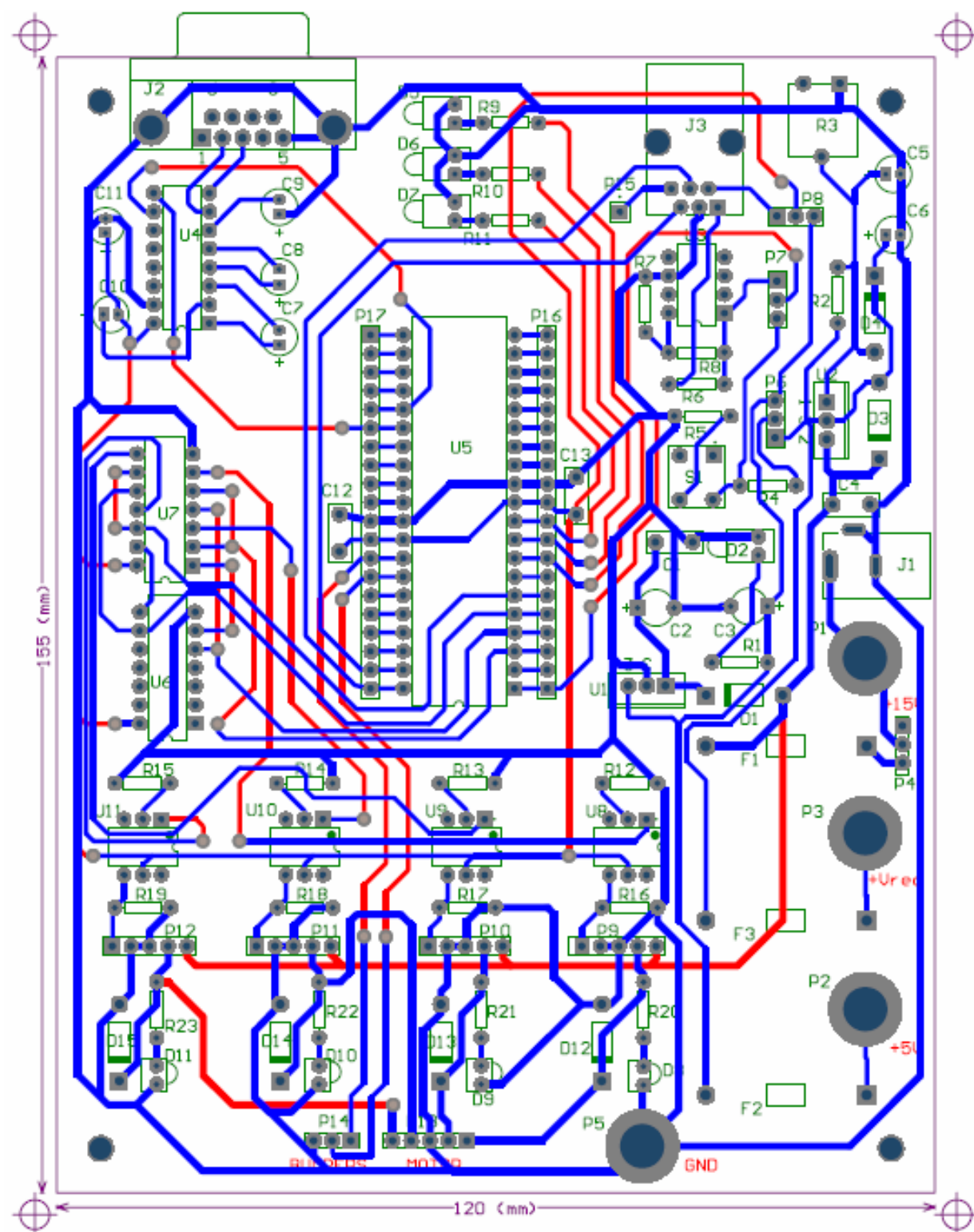


Fig. 78: Diagrama general del hardware con los dispositivos y las pistas (en rojo: superior, en azul: inferior)

Anexo F.: Data Sheets

Debido al gran número de componentes utilizados y a la extensión de las hojas de especificaciones de algunos integrados, se redirecciona a la web del fabricante para obtener los *Data Sheet* de cada componente.

Tabla 18: Links de los *Data sheet* de los integrados utilizados.

Componente	Link del <i>Data Sheet</i>
PIC16F727	http://ww1.microchip.com/downloads/en/DeviceDoc/41341E.pdf
1N4007	http://www.vishay.com/docs/88503/1n4001.pdf
1N4933	http://www.vishay.com/docs/88508/1n4933.pdf
UC3710	http://focus.ti.com/lit/ds/symlink/uc3710.pdf
L7805	http://www.st.com/stonline/products/literature/ds/2143/l7805c.pdf
LM317	http://www.st.com/stonline/products/literature/ds/2154/lm317.pdf
TL431	http://focus.ti.com/lit/ds/symlink/tl431.pdf
MAX232	http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf
SN74LS04	http://focus.ti.com/lit/ds/symlink/sn74ls04.pdf
SN74LS08	http://focus.ti.com/lit/ds/symlink/sn74ls08.pdf
4N27	http://www.vishay.com/docs/83725/4n25.pdf

De igual forma se redirecciona a la web del fabricante para obtener las especificaciones técnicas de los instrumentos de medida.

Tabla 19: Links de los *Data Sheet* de los instrumentos de medida.

Instrumento de Medida	Link del <i>Data Sheet</i>
Agilent 8510C	http://cp.literature.agilent.com/litweb/pdf/5091-8484E.pdf

Panasonic

Stepping Motor **PM TYPE** **55SPM25**

■ Characteristics

B. Constant Current Driving

